US011226975B2

(12) United States Patent
Patthak et al.

(10) **Patent No.: US 11,226,975 B2**

(45) **Date of Patent: Jan. 18, 2022**

(54) **METHOD AND SYSTEM FOR IMPLEMENTING MACHINE LEARNING CLASSIFICATIONS**

(71) Applicant: **ORACLE INTERNATIONAL CORPORATION**, Redwood Shores, CA (US)

(72) Inventors: **Anindya Chandra Patthak**, San Jose, CA (US); **Gregory Michael Ferrar**, Santa Cruz, CA (US)

(73) Assignee: **Oracle International Corporation**, Redwood Shores, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 538 days.

(21) Appl. No.: **15/089,226**

(22) Filed: **Apr. 1, 2016**

(65) **Prior Publication Data**

US 2016/0292592 A1 Oct. 6, 2016

Related U.S. Application Data

(60) Provisional application No. 62/142,987, filed on Apr. 3, 2015.

(51) **Int. Cl.**

G06F 11/00 (2006.01)
G06F 11/07 (2006.01)
(Continued)

(52) **U.S. Cl.**

CPC ***G06F 16/248*** (2019.01); ***G06F 3/04842*** (2013.01); ***G06F 9/44505*** (2013.01);
(Continued)

(58) **Field of Classification Search**

CPC .. G06F 11/00; G06F 11/0766; G06F 11/0775; G06F 11/3006; G06F 11/3072;
(Continued)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,832,511	A	11/1998	Beck et al.
6,725,262	B1	4/2004	Choquier et al.

(Continued)

FOREIGN PATENT DOCUMENTS

CN	1352768	A	6/2002
CN	101267352	B	5/2011

(Continued)

OTHER PUBLICATIONS

Li, Weixi "Automatic Log Analysis Using Machine Learning: Awesome Automatic Log Analysis Version 2.0", Uppsala Universitet, Nov. 2013. (Year: 2013).*

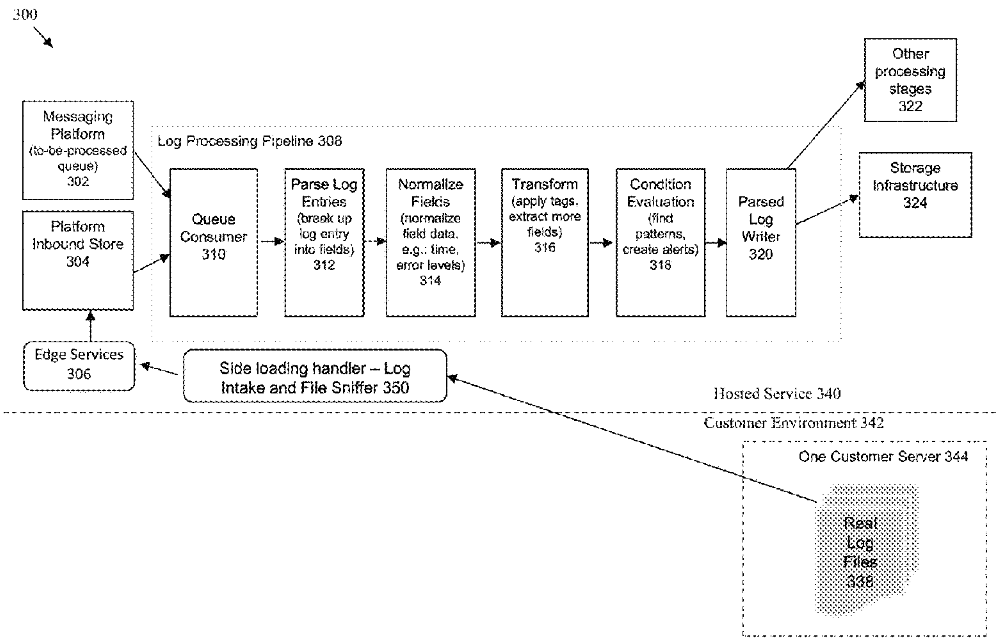
(Continued)

Primary Examiner — Miranda M Huang
Assistant Examiner — Robert Lewis Kulp
(74) *Attorney, Agent, or Firm* — Invoke

(57) **ABSTRACT**

Disclosed is a system, method, and computer program product for implementing a log analytics method and system that can configure, collect, and analyze log records in an efficient manner. Machine learning-based classification can be performed to classify logs. This approach is used to group logs automatically using a machine learning infrastructure.

19 Claims, 53 Drawing Sheets


300
Messaging Platform (to-be-processed queue) 302
Platform Inbound Store 304
Edge Services 306
Log Processing Pipeline 308
Queue Consumer 310
Parse Log Entries (break up log entry into fields) 312
Normalize Fields (normalize field data, e.g.: time, error levels) 314
Transform (apply tags, extract more fields) 316
Condition Evaluation (find patterns, create alerts) 318
Parsed Log Writer 320
Other processing stages 322
Storage Infrastructure 324
Side loading handler – Log Intake and File Sniffer 350
Hosted Service 340
Customer Environment 342
One Customer Server 344
Real Log Files 338

(51) **Int. Cl.**

G06F 11/30	(2006.01)
G06F 16/21	(2019.01)
G06F 16/22	(2019.01)
G06F 16/2455	(2019.01)
G06F 16/248	(2019.01)
G06F 16/35	(2019.01)
G06F 16/84	(2019.01)
G06F 3/0484	(2013.01)
G06F 9/445	(2018.01)
G06F 9/54	(2006.01)
G06F 40/16	(2020.01)
G06N 20/00	(2019.01)
H04L 12/24	(2006.01)
H04L 12/26	(2006.01)
G06F 40/205	(2020.01)

(52) **U.S. Cl.**

CPC ***G06F 9/542*** (2013.01); ***G06F 11/00*** (2013.01); ***G06F 11/0766*** (2013.01); ***G06F 11/0775*** (2013.01); ***G06F 11/3006*** (2013.01); ***G06F 11/3072*** (2013.01); ***G06F 11/3086*** (2013.01); ***G06F 16/21*** (2019.01); ***G06F 16/2228*** (2019.01); ***G06F 16/2455*** (2019.01); ***G06F 16/353*** (2019.01); ***G06F 16/84*** (2019.01); ***G06F 40/16*** (2020.01); ***G06F 40/205*** (2020.01); ***G06N 20/00*** (2019.01); ***H04L 41/5074*** (2013.01); ***H04L 43/04*** (2013.01); *H04L 41/145* (2013.01)

(58) **Field of Classification Search**

CPC .. G06F 11/3086; G06F 16/21; G06F 16/2228; G06F 16/2455; G06F 9/44505; G06F 9/542; G06N 20/00; G06N 99/005; H04L 41/145; H04L 41/5074

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,738,911	B2	5/2004	Hayes
6,910,183	B2	6/2005	Maier et al.
7,111,075	B2	9/2006	Pankovcin et al.
7,155,514	B1	12/2006	Milford
7,523,433	B1	4/2009	Anderson
7,653,633	B2	1/2010	Villella et al.
7,822,850	B1	10/2010	Brikman et al.
7,844,999	B1	11/2010	Agmlar-Macias et al.
8,041,683	B1	10/2011	Korolev et al.
8,612,377	B2	12/2013	Beg et al.
8,620,928	B1	12/2013	Walton
8,832,125	B2	9/2014	Boctor
9,092,625	B1	7/2015	Kashyap et al.
9,262,519	B1 *	2/2016	Saurabh G06F 16/285
9,292,328	B2	3/2016	Pratt et al.
2002/0138762	A1	9/2002	Horne
2004/0254919	A1	12/2004	Giuseppini
2004/0261055	A1	12/2004	Bertelrud et al.
2005/0027858	A1	2/2005	Sloth et al.
2005/0044075	A1	2/2005	Steere et al.
2005/0228885	A1	10/2005	Winfield et al.
2006/0136177	A1	6/2006	Patanian
2006/0195297	A1	8/2006	Kubota et al.
2006/0195731	A1	8/2006	Patterson et al.
2006/0214963	A1	9/2006	Komatsu
2008/0005265	A1	1/2008	Miettinen et al.
2008/0155103	A1	6/2008	Bailey
2008/0215546	A1	9/2008	Baum et al.
2009/0089252	A1	4/2009	Galitsky et al.
2009/0119307	A1	5/2009	Braun et al.
2009/0249250	A1	10/2009	Gajula et al.
2010/0115010	A1	5/2010	Anderson et al.
2011/0246826	A1	10/2011	Hsieh et al.
2012/0005542	A1	1/2012	Petersen et al.
2012/0117079	A1	5/2012	Baum et al.
2012/0124047	A1 *	5/2012	Hubbard G06F 17/30637 707/737
2012/0278872	A1	11/2012	Woelfel et al.
2013/0054402	A1	2/2013	Asherman et al.
2013/0227352	A1	8/2013	Kumarasamy et al.
2013/0282739	A1	10/2013	Anderson et al.
2014/0089744	A1	3/2014	Oshiro
2014/0157370	A1	6/2014	Plattner et al.
2014/0289428	A1	9/2014	Walter et al.
2014/0304197	A1	10/2014	Jaiswal et al.
2015/0149480	A1	5/2015	Swan et al.
2015/0154192	A1 *	6/2015	Lysne G06N 5/00 707/748
2015/0293920	A1	10/2015	Kanjirathinkal et al.
2015/0379052	A1	12/2015	Agarwal et al.
2016/0019286	A1	1/2016	Bach et al.
2016/0034510	A1	2/2016	Gukal
2016/0041894	A1	2/2016	Reid et al.
2016/0092427	A1 *	3/2016	Bittmann G06F 40/30 704/9
2016/0092558	A1	3/2016	Ago et al.
2016/0224570	A1	8/2016	Sharp et al.
2016/0246849	A1	8/2016	Frampton et al.
2016/0247205	A1	8/2016	Ziliacus et al.
2016/0253425	A1	9/2016	Stoops et al.
2016/0292263	A1	10/2016	Ferrar
2016/0292592	A1	10/2016	Patthak et al.
2016/0371363	A1	12/2016	Muro et al.

FOREIGN PATENT DOCUMENTS

CN	102164050	A	8/2011
CN	103412924	A	11/2013
CN	105138593	A	12/2015
WO	2012/031259	A1	3/2012

OTHER PUBLICATIONS

Xu, Wei, "Detecting Large-Scale System Problems by Mining Console Logs," SOSP'09, Oct. 11-14, 2009, pp. 117-131 (Year: 2009).*

Ning et al ("HLAer: a System for Heterogeneous Log Analysis", "HLAer: a System for Heterogeneous Log Analysis," in SDM Workshop on Heterogeneous Machine Learning, 2014, pp. 1-22) (Year: 2014).*

Han et al("Centroid-Based Document Classification: Analysis and Experimental Results", In: Zighed D.A., Komorowski J., Żytkow J. (eds) Principles of Data Mining and Knowledge Discovery. PKDD 2000. Lecture Notes in Computer Science, vol. 1910. Springer, 2000, pp. 1-8) (Year: 2000).*

T. G. Dietterich ("Ensemble methods in machine learning", In Multiple Classifier System, Springer, pp. 1-15, 2000) (Year: 2000).*

Xu et al ("Detecting Large-Scale System Problems by Mining Console Logs", SOSP'09, Oct. 11-14, 2009, pp. 117-131) (Year: 2009).*

Marco Lui ("Generalized Language Identification", Phd Thesis, Department of Computing and Information Systems, The University of Melbourne, Jul. 2014, pp. 1-326) (Year: 2014).*

Malmasi et al ("NLI Shared Task 2013: MQ Submission", Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications, pp. 124-133, Atlanta, Georgia, Jun. 13, 2013) (Year: 2013).*

Hayta et al ("Language Identification Based on N-Gram Feature Extraction Method by Using Classifiers", IU-JEEE vol. 13(2), 2013, pp. 1-10) (Year: 2013).*

Selamat et al. ("Arabic Script Web Document Language Identifications Using Neural Network" iiWAS, 2007, pp. 329-338) (Year: 2007).*

International Search Report and Written Opinion dated Jul. 7, 2016 for corresponding PCT Patent Application No. PCT/US2016/025739.

(56) References Cited

OTHER PUBLICATIONS

Wei Xu, et al., “Detecting large-scale system problems by mining console logs”, Proceedings of the ACM SIGOPS 22nd Symposium on Operation Systems Principles, SOSP ’09, Jan. 1, 2009, p. 117, XP055254995, New York, New York, USA.

Meiyappan Nagappan, et al., “Abstracting log lines to log event types for mining software system logs”, Mining Software Repositories (MSR), 2010, 7th IEEE Working Conference on, IEEE, Piscataway, NJ, USA, May 2, 2010, pp. 114-117, XP031675571.

Hongyong Yu, et al., “Mass log data processing and mining based on Hadoop and cloud computing” Computer Science & Education (ICCSE), 2010, 7th International Conference on, IEEE, Jul. 14, 2012, pp. 197-202, XP032232566.

School of Haskell, “Parsing Log Files in Haskell”, Feb. 1, 2015, 22 pages.

Loggly, “Automated Parsing Log Types”, Support Center, Apr. 8, 2015, 15 pages https://wvw.loggly.com/docs/automated-parsing/.

“Log monitoring/analysis” May 13, 2014, 4 pages http://ossec-docs.readthedocs.org/en/latest/manual/monitoring/.

Scalyr, “Parsing Logs”, Apr. 19, 2015, 9 pages https://www.scalyr.com/help/parsing-logs/.

DataDOG DOCS, “Log Parsing in the Agent”, Jul. 20, 2013, 3 pages http://docs.datadoghq.com/guides/logs/.

Gamuts Software, “Log File Parsers”, Mar. 5, 2015, 1 page http://www.gamutsoftware.com/index.php/help/logfileconfiguration/.

Logentries, “Tags and Alerts”, Jul. 6, 2015, 5 pages https://logentries.com/doc/setup-tags-alerts/.

William Lam, “How To Add A Tag (Log prefix) To Syslog Entries”, May 7, 2013, 4 pages.

Loggly, “Tags”, Support Center, Apr. 30, 2015, 5 pages https://www.loggly.com/docs/tags/.

Loggly, “Tag Your Sources for More Focused Searching” Sep. 26, 2015, 5 pages https://www.loggly.com/blog/log-management-tags-searching/.

F1 score, Wikipedia, https://en.wikipedia.org/wiki/F1 score, retrieved on Aug. 2, 2018.

Loggly, “Automated Parsing Log Types”, Support Center, Apr. 8, 2015, 15 pages https://wvw.loagly.com/docs/automated-parsing/.

Tf-idf, Wikipedia, https://en.wikipedia.org/wiki/Tf-idf, retrieved on Aug. 9, 2018.

Trevino, Introduction to K-means Clustering, https://www.datascience.com/blog/k-means-clustering, Jun. 12, 2016.

Yarowsky algorithm, Wikipedia, https://en.wikipedia.org/wiki/Yarowsky_algorithm, retrieved on Sep. 12, 2018.

Cohn et al., “Audio De-identification: A New Entity Recognition Task,” NAACL-HLT, 2019, 8 pages.

Cumby et al., “A Machine Learning Based System for Semi-Automatically Redacting Documents,” Proceedings of the Twenty-Third Innovative Applications of Artificial Intelligence Conference, 2011, pp. 1628-1635.

Demoncourt et al., “De-identification of patient notes with recurrent neural networks,” Journal of the American Medical Informatics Association, JAMIA, vol. 24, No. 3, 2016, pp. 596-606.

Irmak et al., “A Scalable Machine-Learning Approach for Semi-Structured Named Entity Recognition,” WWW ’10 Proceedings of the 19th international conference on World wide web, Apr. 26-30, 2010, pp. 461-470.

Liu et al., “De-identification of clinical notes via recurrent neural network and conditional random field,” Journal of Biomedical Informatics, vol. 75, 2017, pp. S34-S42.

Mamou et al., “Term Set Expansion based NLP Architect by Intel AI Lab,” Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (System Demonstrations), 2018, pp. 19-24.

Qiu et al., “Learning Word Representation Considering Proximity and Ambiguity,” AAAI’14 Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014, pp. 1572-1578.

Sahlgren et al., “Learning Representations for Detecting Abusive Language,” Proceedings of the Second Workshop on Abusive Language Online (ALW2), 2018, pp. 115-123.

Shin et al., “Electronic Medical Records privacy preservation through k-anonymity clustering method,” The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems, 2012, 1119-1124.

Towards Automated Log Parsing for Large-Scale Log Data Analysis Pinjia He;Jieming Zhu;Shilin He;Jian Li;Michael R. Lyu IEEE Transactions on Dependable and Secure Computing, 933-944 (Year: 2019).

* cited by examiner

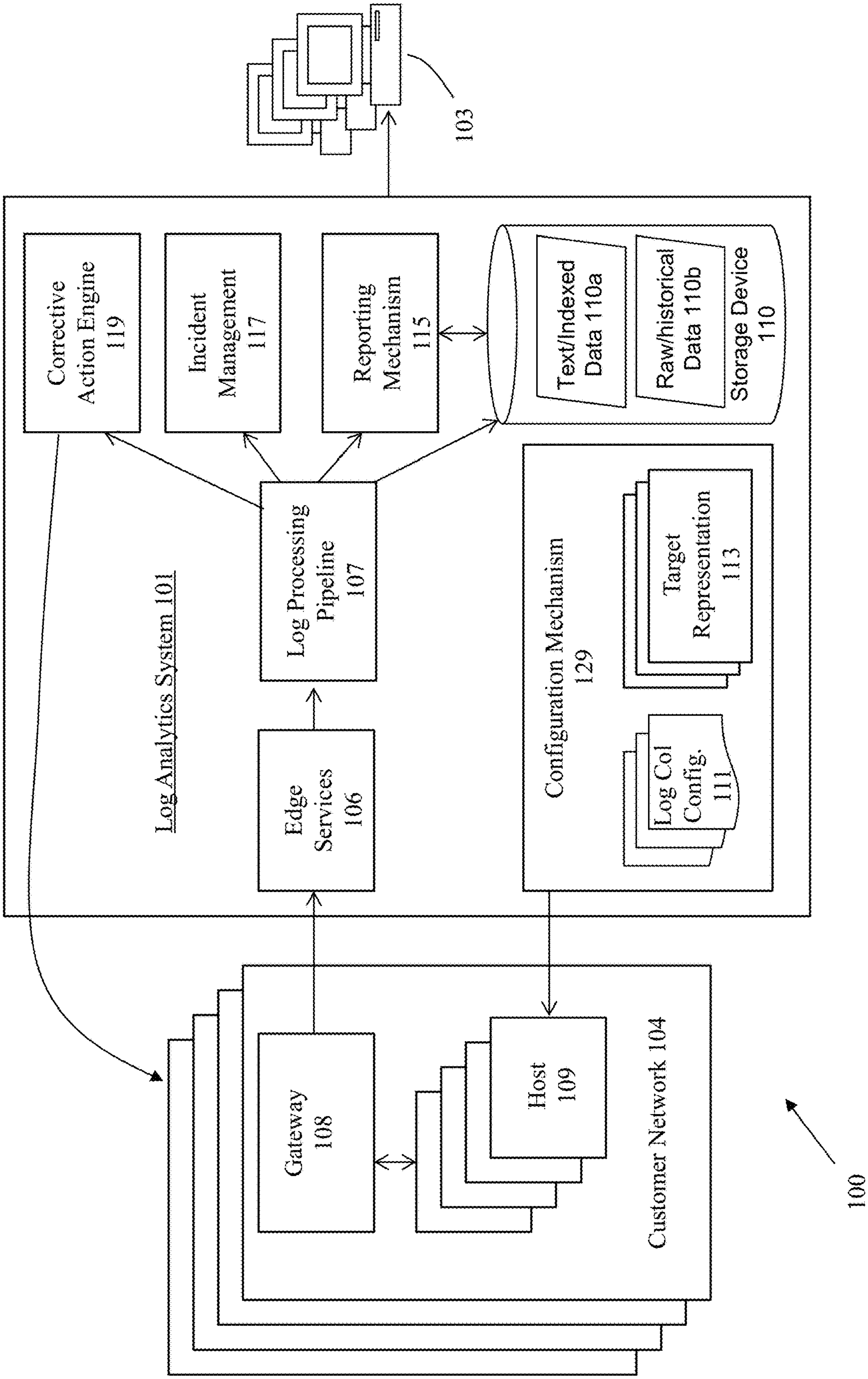

103
Log Analytics System 101
Corrective Action Engine 119
Incident Management 117
Reporting Mechanism 115
Text/Indexed Data 110a
Raw/historical Data 110b
Storage Device 110
Log Processing Pipeline 107
Edge Services 106
Configuration Mechanism 129
Target Representation 113
Log Col Config. 111
Gateway 108
Host 109
Customer Network 104
100

Fig. 1A

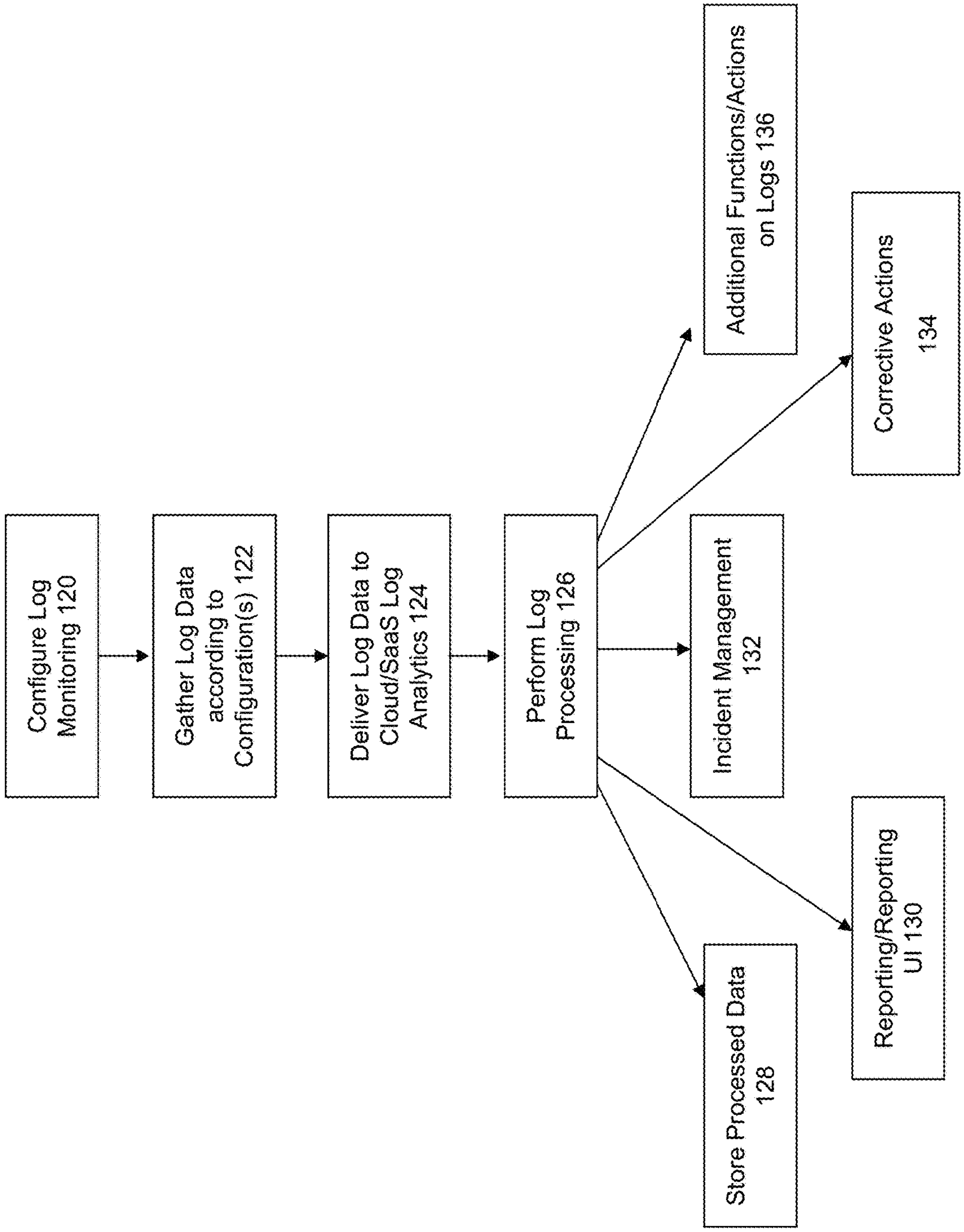

Configure Log Monitoring 120
Gather Log Data according to Configuration(s) 122
Deliver Log Data to Cloud/SaaS Log Analytics 124
Perform Log Processing 126
Store Processed Data 128
Reporting/Reporting UI 130
Incident Management 132
Corrective Actions 134
Additional Functions/Actions on Logs 136

Fig. 1B

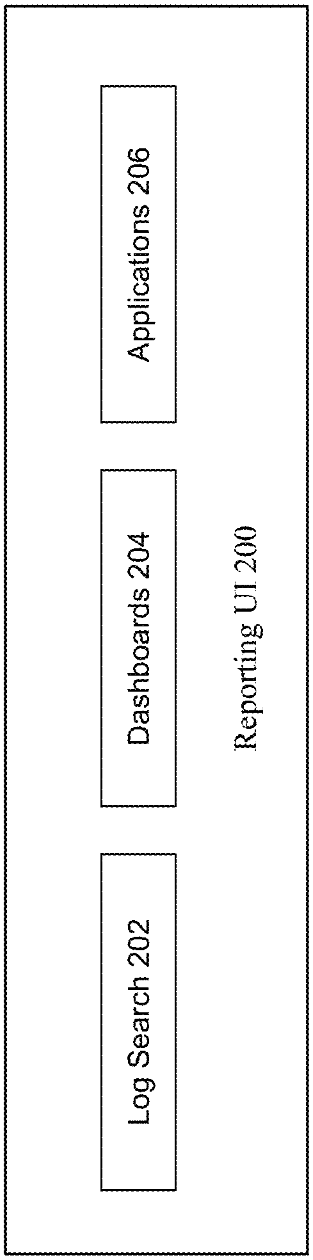

Applications 206
Dashboards 204
Log Search 202
Reporting UI 200

Fig. 2

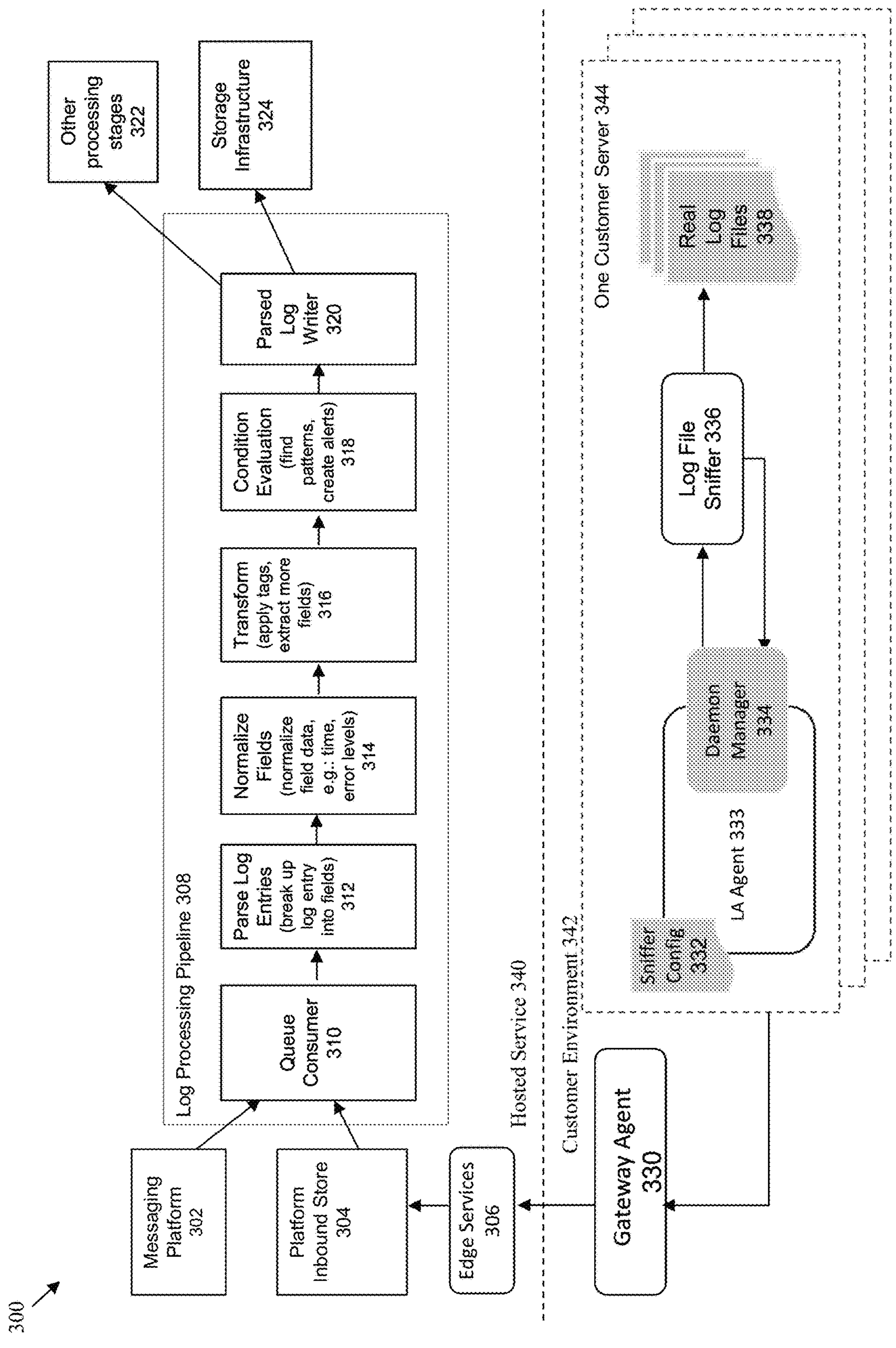

300
Messaging Platform 302
Platform Inbound Store 304
Edge Services 306
Log Processing Pipeline 308
Queue Consumer 310
Parse Log Entries (break up log entry into fields) 312
Normalize Fields (normalize field data, e.g.: time, error levels) 314
Transform (apply tags, extract more fields) 316
Condition Evaluation (find patterns, create alerts) 318
Parsed Log Writer 320
Other processing stages 322
Storage Infrastructure 324
Hosted Service 340
Customer Environment 342
Gateway Agent 330
Sniffer Config 332
LA Agent 333
Daemon Manager 334
Log File Sniffer 336
Real Log Files 338
One Customer Server 344

Fig. 3A

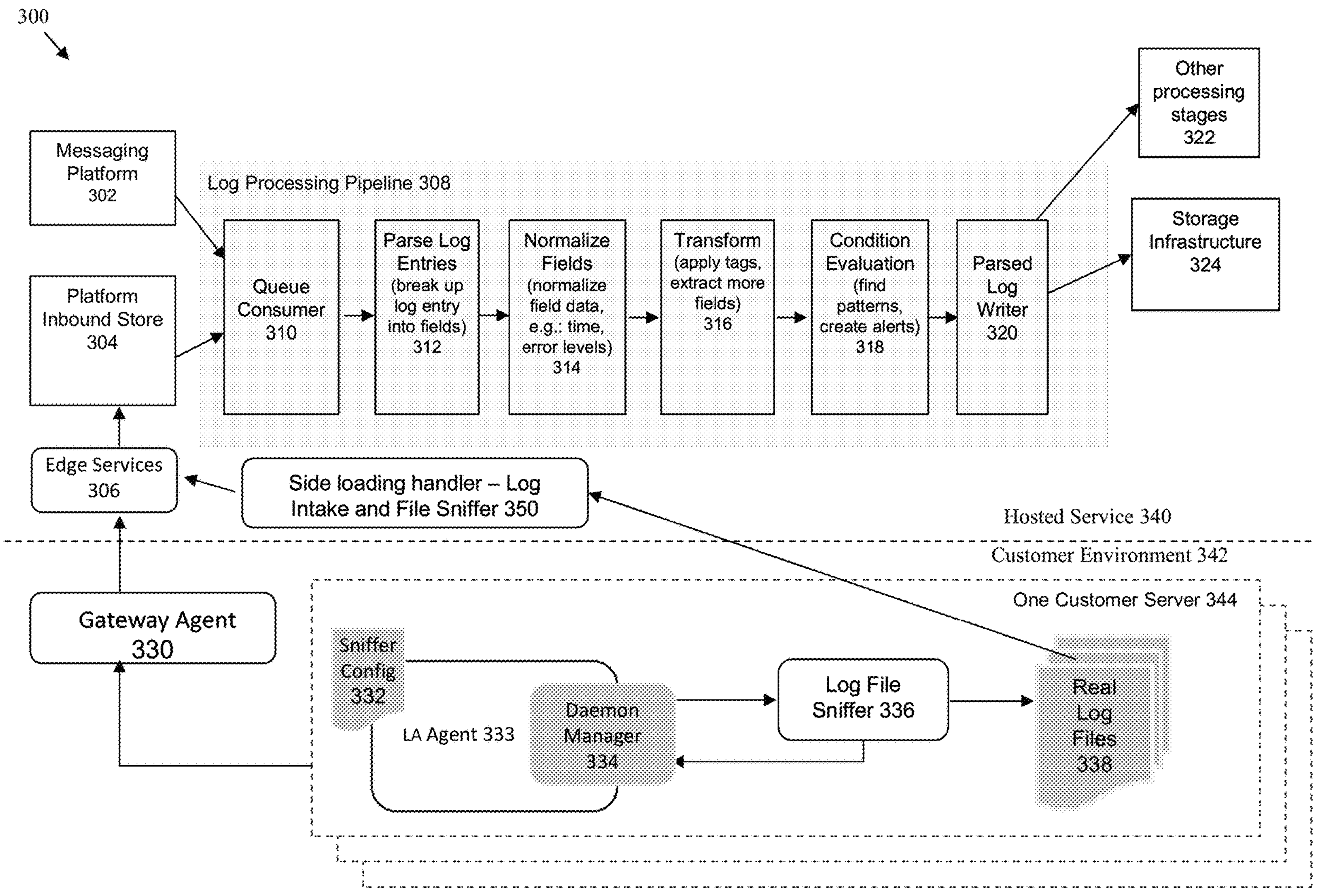

300
Messaging Platform 302
Platform Inbound Store 304
Log Processing Pipeline 308
Queue Consumer 310
Parse Log Entries (break up log entry into fields) 312
Normalize Fields (normalize field data, e.g.: time, error levels) 314
Transform (apply tags, extract more fields) 316
Condition Evaluation (find patterns, create alerts) 318
Parsed Log Writer 320
Other processing stages 322
Storage Infrastructure 324
Edge Services 306
Side loading handler – Log Intake and File Sniffer 350
Hosted Service 340
Customer Environment 342
One Customer Server 344
Gateway Agent 330
Sniffer Config 332
LA Agent 333
Daemon Manager 334
Log File Sniffer 336
Real Log Files 338

Fig. 3B

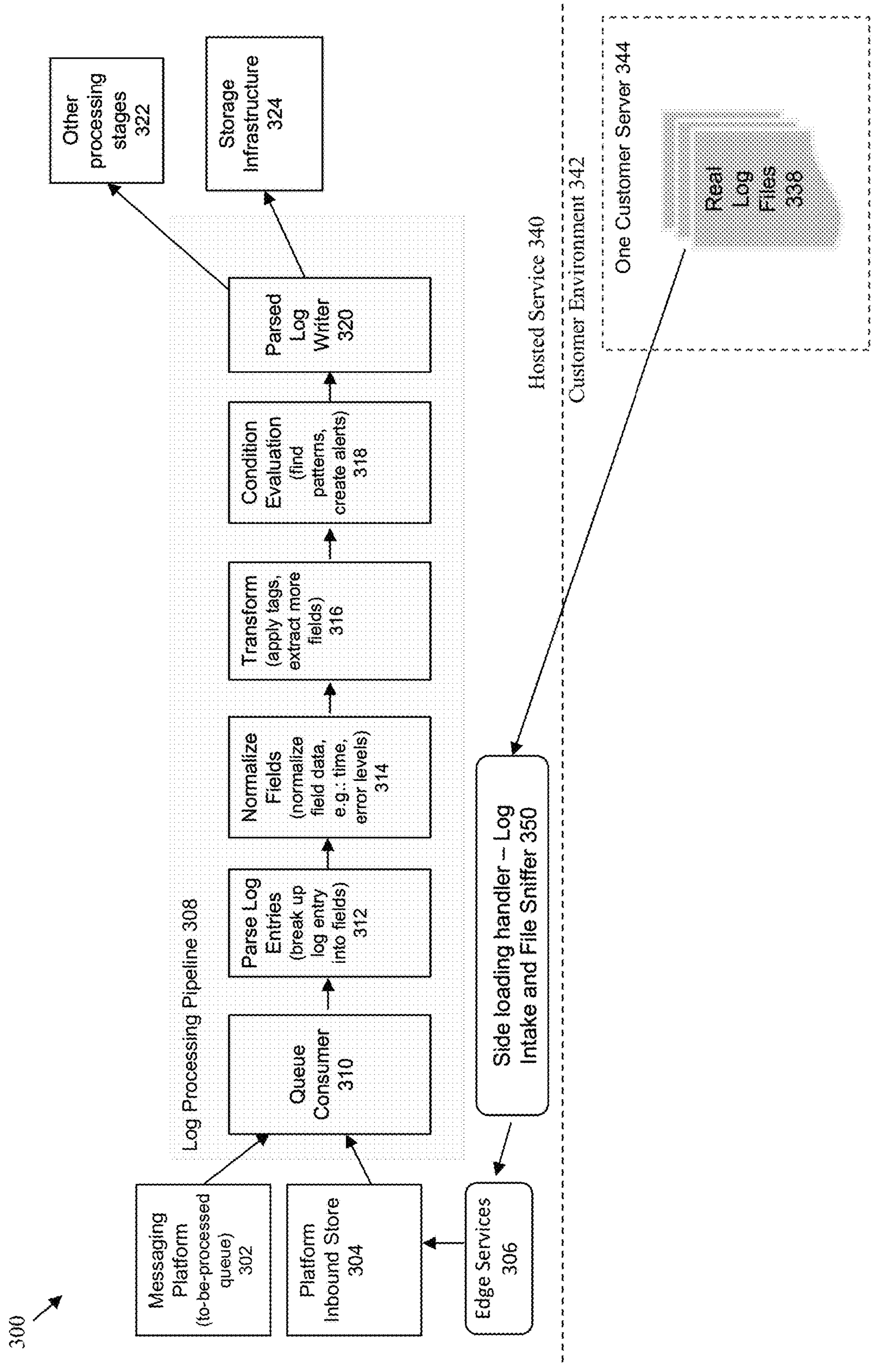

300
Messaging Platform (to-be-processed queue) 302
Platform Inbound Store 304
Edge Services 306
Log Processing Pipeline 308
Queue Consumer 310
Parse Log Entries (break up log entry into fields) 312
Normalize Fields (normalize field data, e.g.: time, error levels) 314
Transform (apply tags, extract more fields) 316
Condition Evaluation (find patterns, create alerts) 318
Parsed Log Writer 320
Other processing stages 322
Storage Infrastructure 324
Side loading handler – Log Intake and File Sniffer 350
Hosted Service 340
Customer Environment 342
One Customer Server 344
Real Log Files 338

Fig. 3C

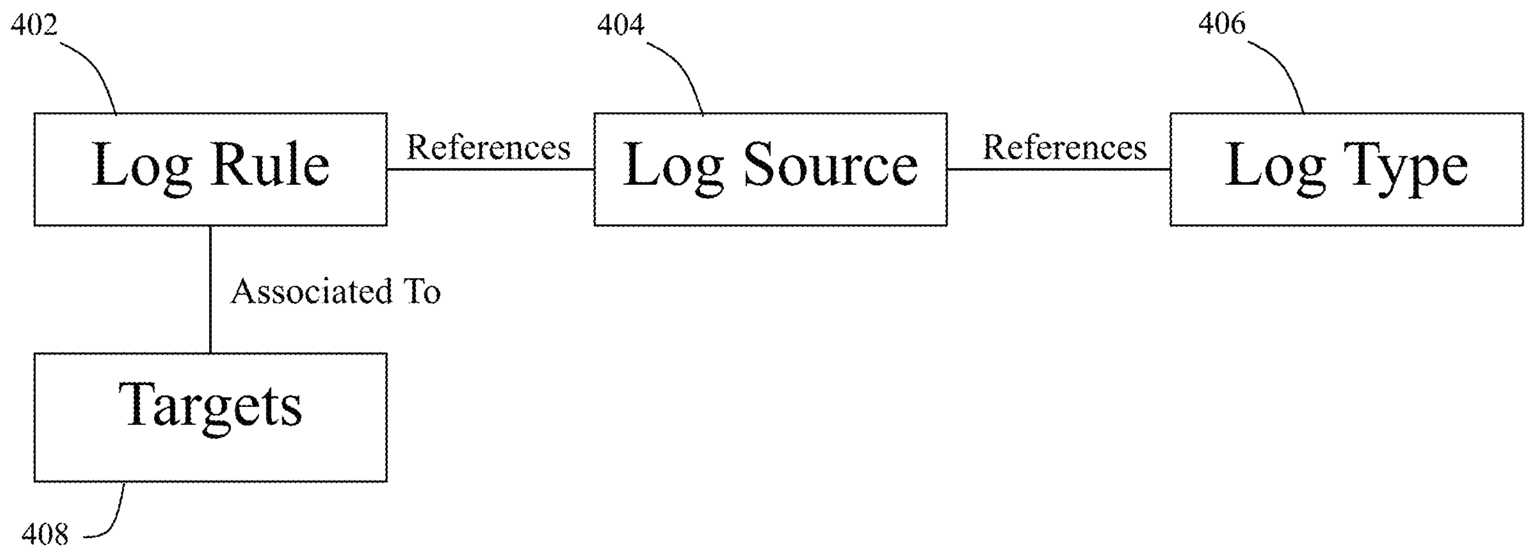

402
404
406
Log Rule
References
Log Source
References
Log Type
Associated To
Targets
408

Fig. 4A

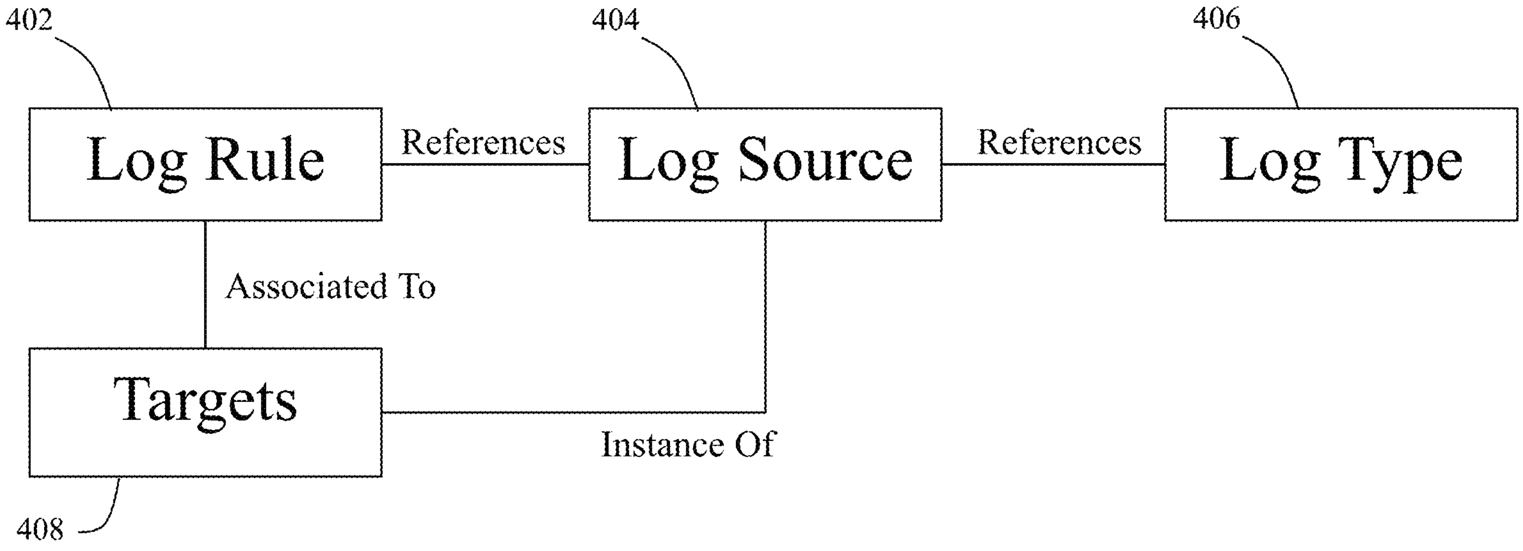

402
404
406
Log Rule
References
Log Source
References
Log Type
Associated To
Targets
Instance Of
408

Fig. 4B

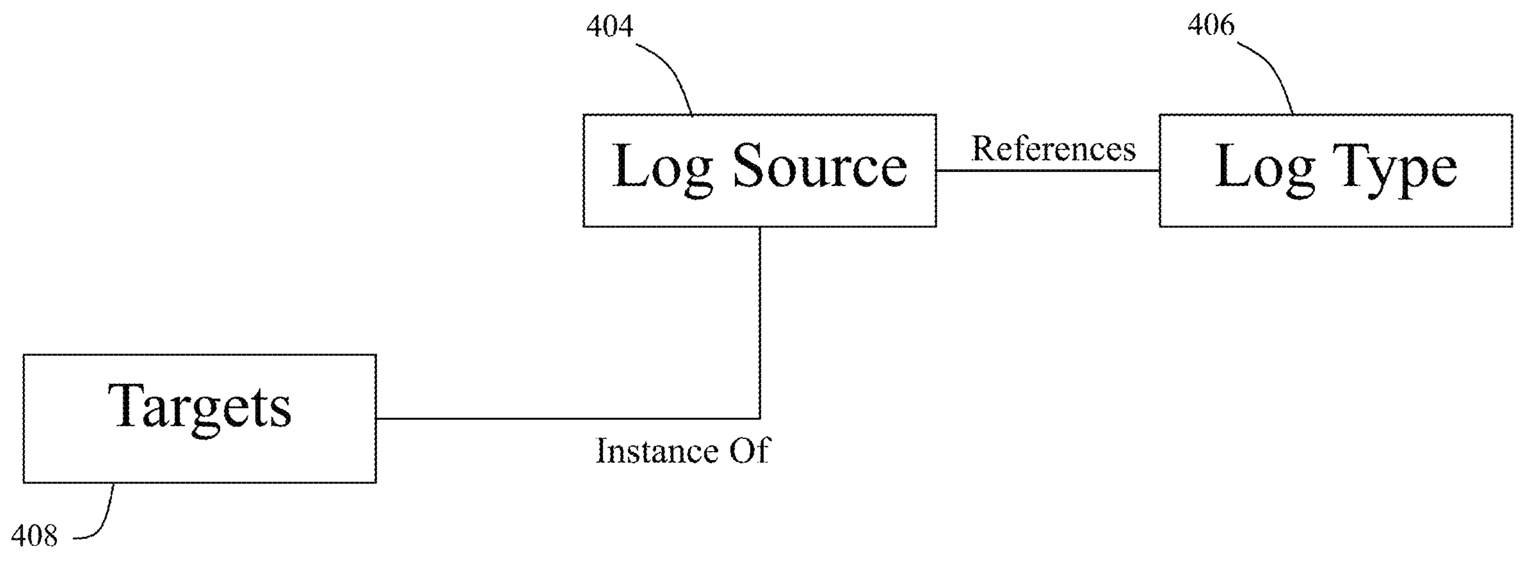

404
406
Log Source
References
Log Type
Targets
Instance Of
408

Fig. 4C

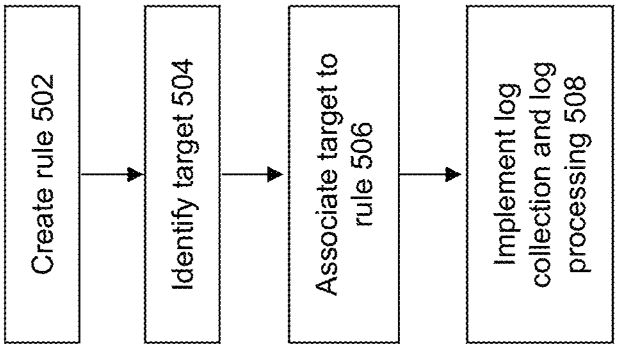

Create rule 502
Identify target 504
Associate target to rule 506
Implement log collection and log processing 508

Fig. 5

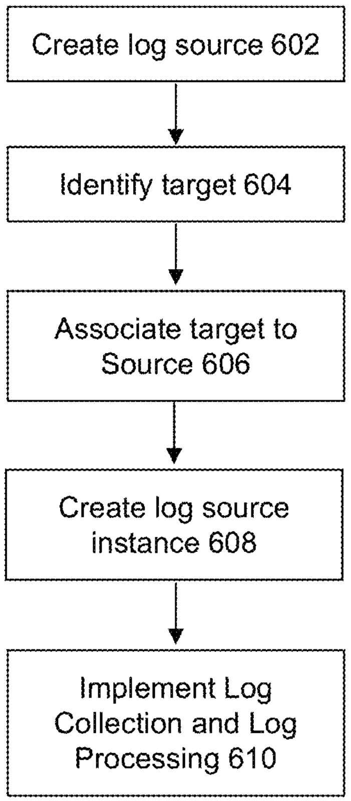

Create log source 602
Identify target 604
Associate target to Source 606
Create log source instance 608
Implement Log Collection and Log Processing 610

Fig. 6

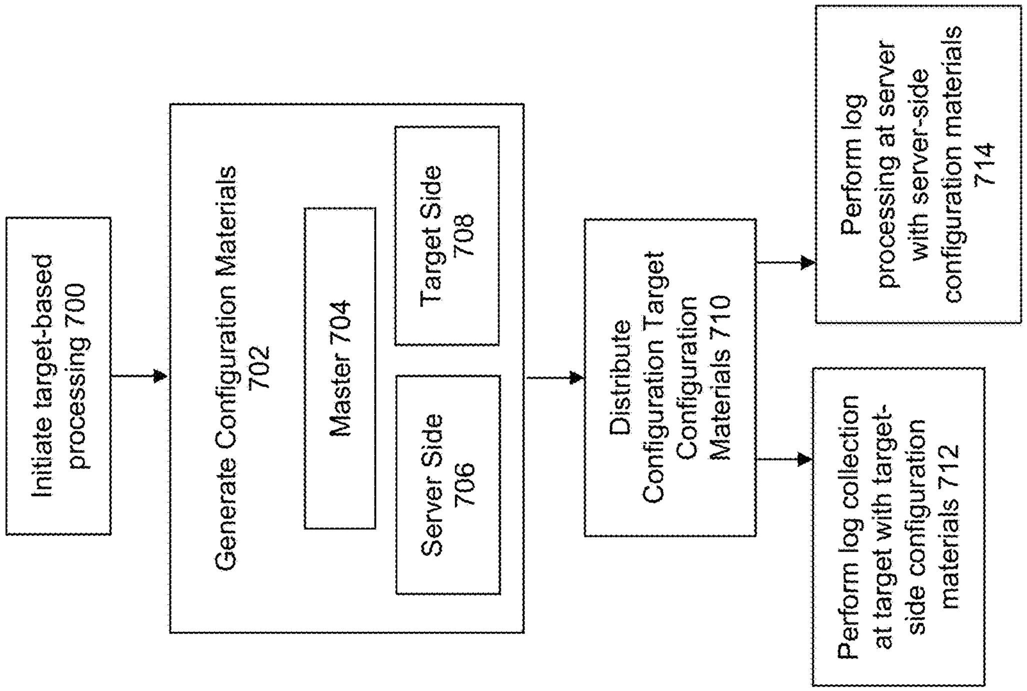

Initiate target-based processing 700
Generate Configuration Materials 702
Master 704
Server Side 706
Target Side 708
Distribute Configuration Target Configuration Materials 710
Perform log collection at target with target-side configuration materials 712
Perform log processing at server with server-side configuration materials 714

Fig. 7

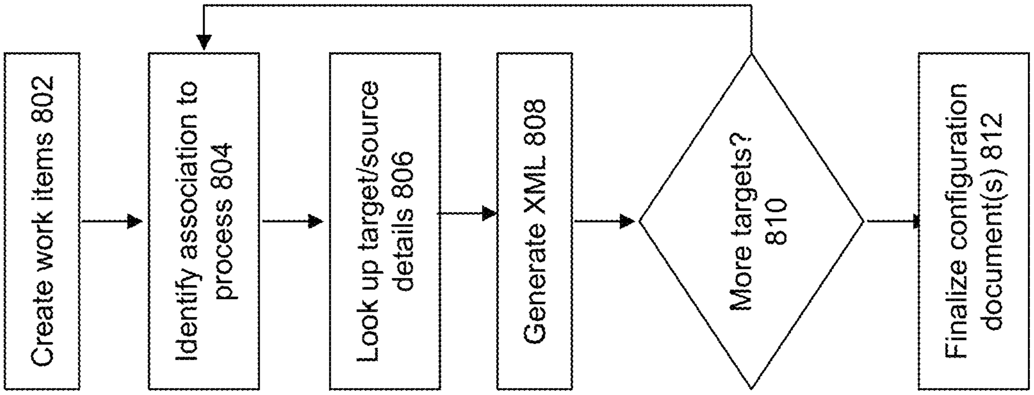
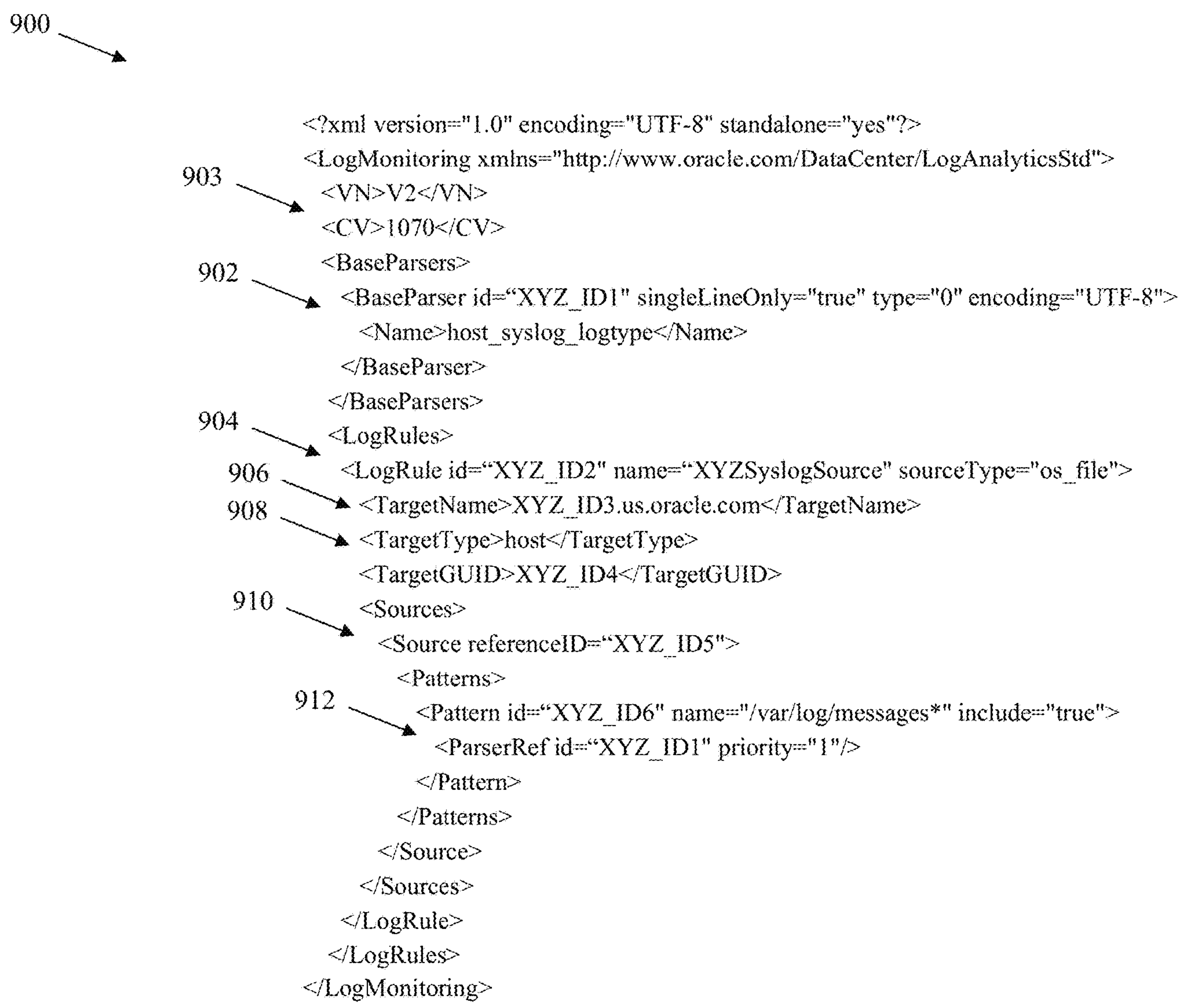

Create work items 802
Identify association to process 804
Look up target/source details 806
Generate XML 808
More targets? 810
Finalize configuration document(s) 812

Fig. 8



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LogMonitoring xmlns="http://www.oracle.com/DataCenter/LogAnalyticsStd">
  <VN>V2</VN>
  <CV>1070</CV>
  <BaseParsers>
    <BaseParser id=“XYZ_ID1" singleLineOnly="true" type="0" encoding="UTF-8">
      <Name>host_syslog_logtype</Name>
    </BaseParser>
   </BaseParsers>
   <LogRules>
    <LogRule id=“XYZ_ID2" name=“XYZSyslogSource" sourceType="os_file">
      <TargetName>XYZ_ID3.us.oracle.com</TargetName>
      <TargetType>host</TargetType>
      <TargetGUID>XYZ_ID4</TargetGUID>
      <Sources>
        <Source referenceID=“XYZ_ID5">
          <Patterns>
            <Pattern id=“XYZ_ID6" name="/var/log/messages*" include="true">
              <ParserRef id=“XYZ_ID1" priority="1"/>
            </Pattern>
          </Patterns>
        </Source>
      </Sources>
    </LogRule>
   </LogRules>
</LogMonitoring>
```

Fig. 9

1000

1003 1001 1002 1004 1006

```
<CV>1070</CV>
<FieldDef name="service" DataType="STRING" MaxSize="4000"/>
<LogSource id="XZY_ID10" name="LinuxSyslogSource" sourceType="os_file"/>
<BaseParser id="XYZ_ID11" singleLineOnly="true" type="0" locale="en_US"
encoding="UTF-8">
  <Name>host_syslog_logtype</Name>
  <Regex>(\S+)\s+(\d+)\s(\d+):(\d+):(\d+)\s(\S+)\s(?:([^:\[]+)(?:\[(\d+)\])?:\s+)?(.+)</Regex>
    <BaseFields>
      <BaseField seq="1" name="timemonthshortname"/>
      <BaseField seq="2" name="timeday"/>
      <BaseField seq="3" name="timehour24"/>
      <BaseField seq="4" name="timeminute"/>
      <BaseField seq="5" name="timesecond"/>
      <BaseField seq="6" name="srvrhostname"/>
      <BaseField seq="7" name="service"/>
      <BaseField seq="8" name="ospid"/>
      <BaseField seq="9" name="msg"/>
    </BaseFields>
</BaseParser>
```

Fig. 10

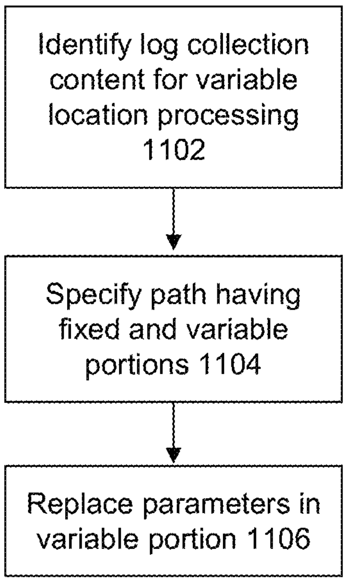

Identify log collection content for variable location processing 1102
Specify path having fixed and variable portions 1104
Replace parameters in variable portion 1106

Fig. 11

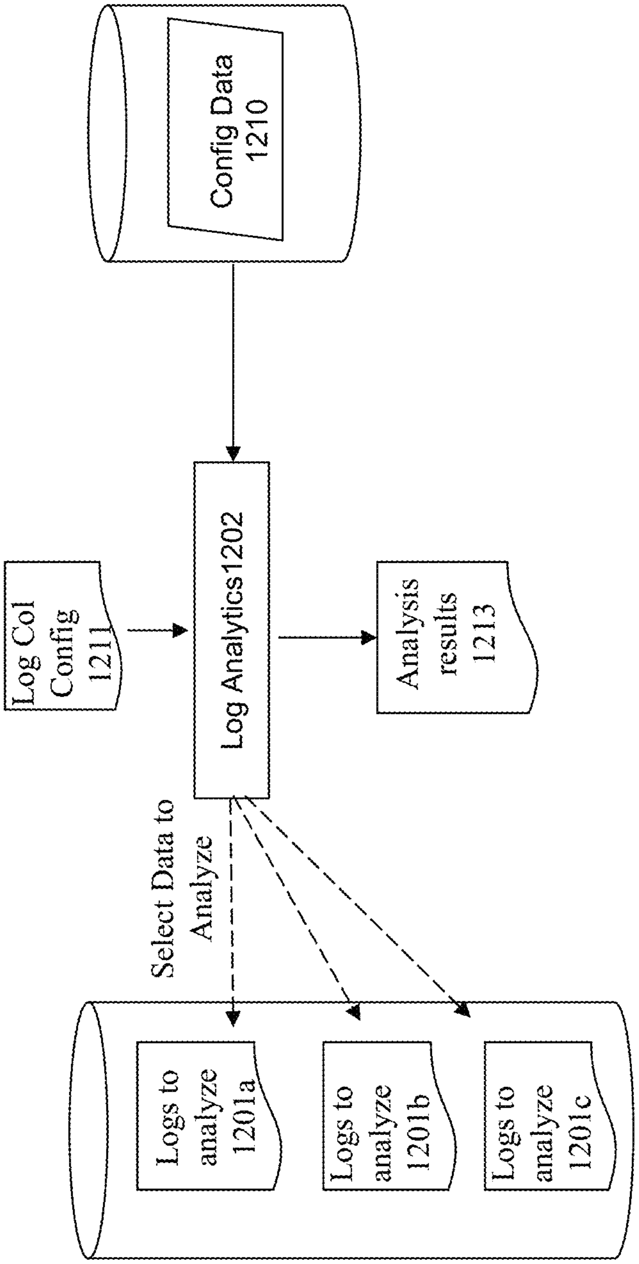
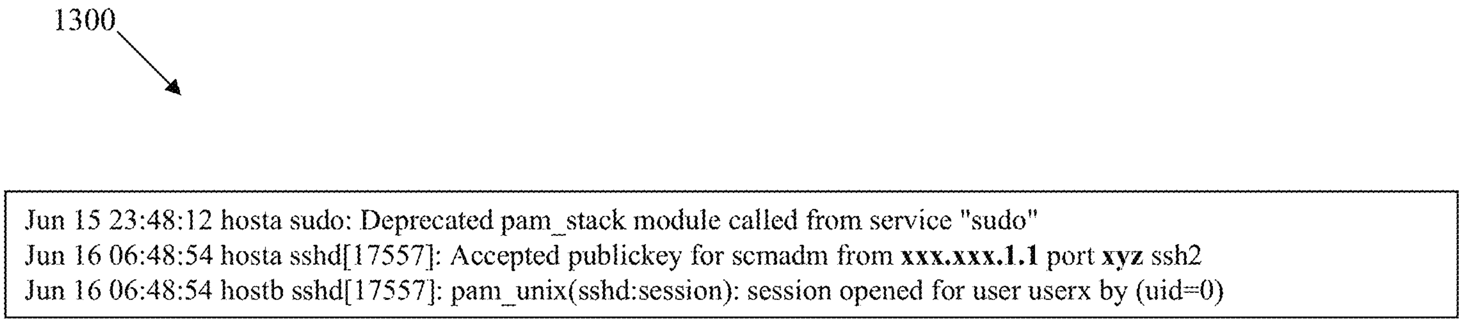
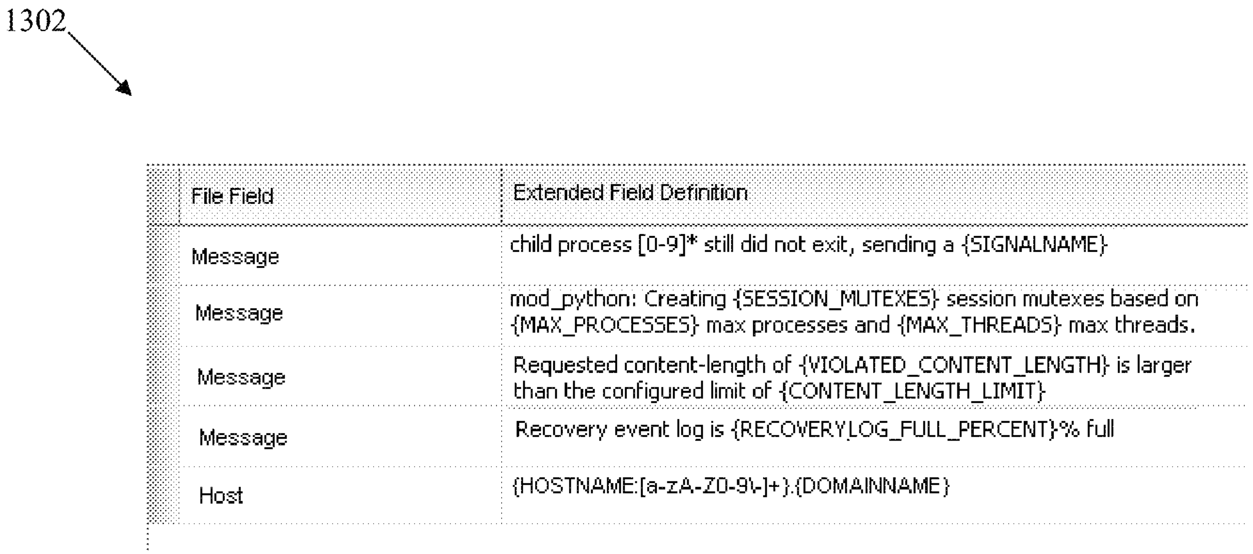

Config Data
1210
Log Col
Config
1211
Log Analytics1202
Analysis
results
1213
Select Data to
Analyze
Logs to
analyze
1201a
Logs to
analyze
1201b
Logs to
analyze
1201c

Fig. 12



```
Jun 15 23:48:12 hosta sudo: Deprecated pam_stack module called from service "sudo"
Jun 16 06:48:54 hosta sshd[17557]: Accepted publickey for scmadm from xxx.xxx.1.1 port xyz ssh2
Jun 16 06:48:54 hostb sshd[17557]: pam_unix(sshd:session): session opened for user userx by (uid=0)
```

Fig. 13



File Field	Extended Field Definition
Message	child process [0-9]* still did not exit, sending a {SIGNALNAME}
Message	mod_python: Creating {SESSION_MUTEXES} session mutexes based on {MAX_PROCESSES} max processes and {MAX_THREADS} max threads.
Message	Requested content-length of {VIOLATED_CONTENT_LENGTH} is larger than the configured limit of {CONTENT_LENGTH_LIMIT}
Message	Recovery event log is {RECOVERY_LOG_FULL_PERCENT}% full
Host	{HOSTNAME:[a-zA-Z0-9\-]+}.{DOMAINNAME}

Fig. 14

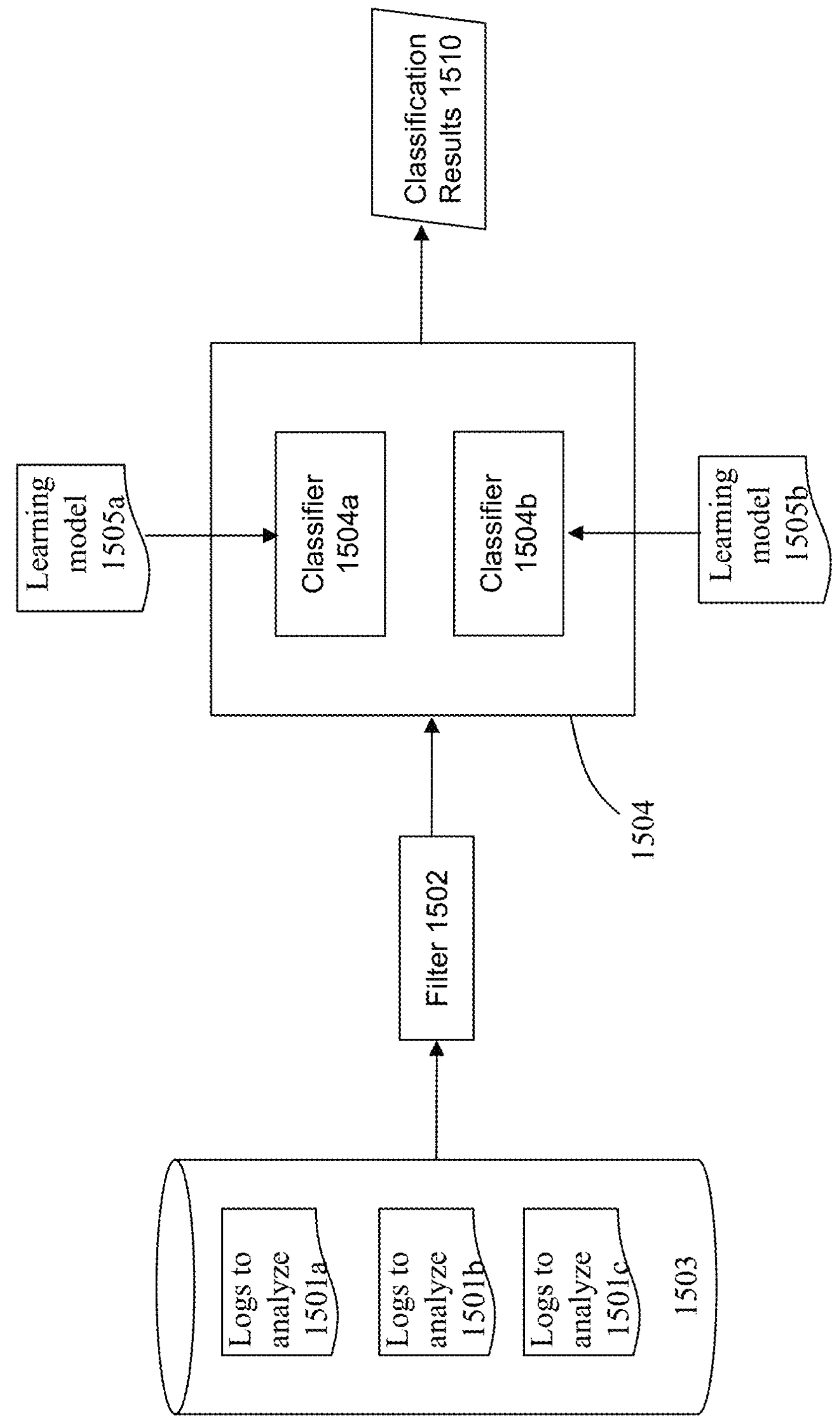

Logs to analyze 1501a
Logs to analyze 1501b
Logs to analyze 1501c
1503
Filter 1502
Learning model 1505a
Classifier 1504a
Classifier 1504b
Learning model 1505b
1504
Classification Results 1510

Fig. 15

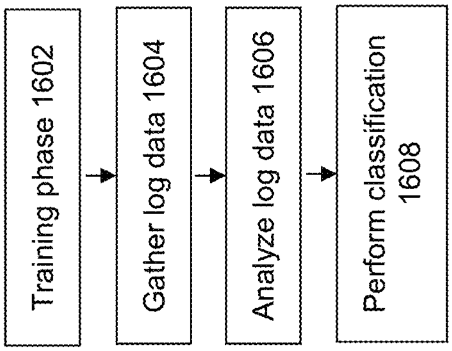

Training phase 1602
Gather log data 1604
Analyze log data 1606
Perform classification 1608

Fig. 16

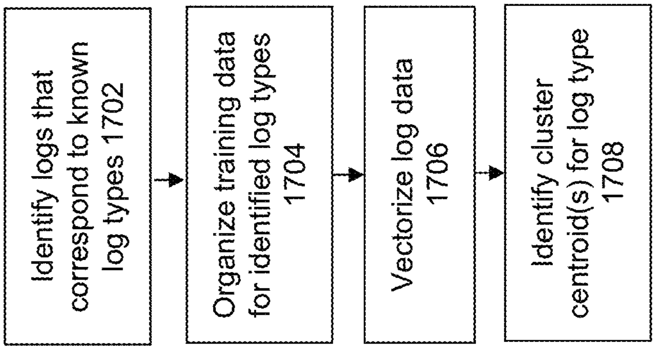

Identify logs that correspond to known log types 1702
Organize training data for identified log types 1704
Vectorize log data 1706
Identify cluster centroid(s) for log type 1708

Fig. 17

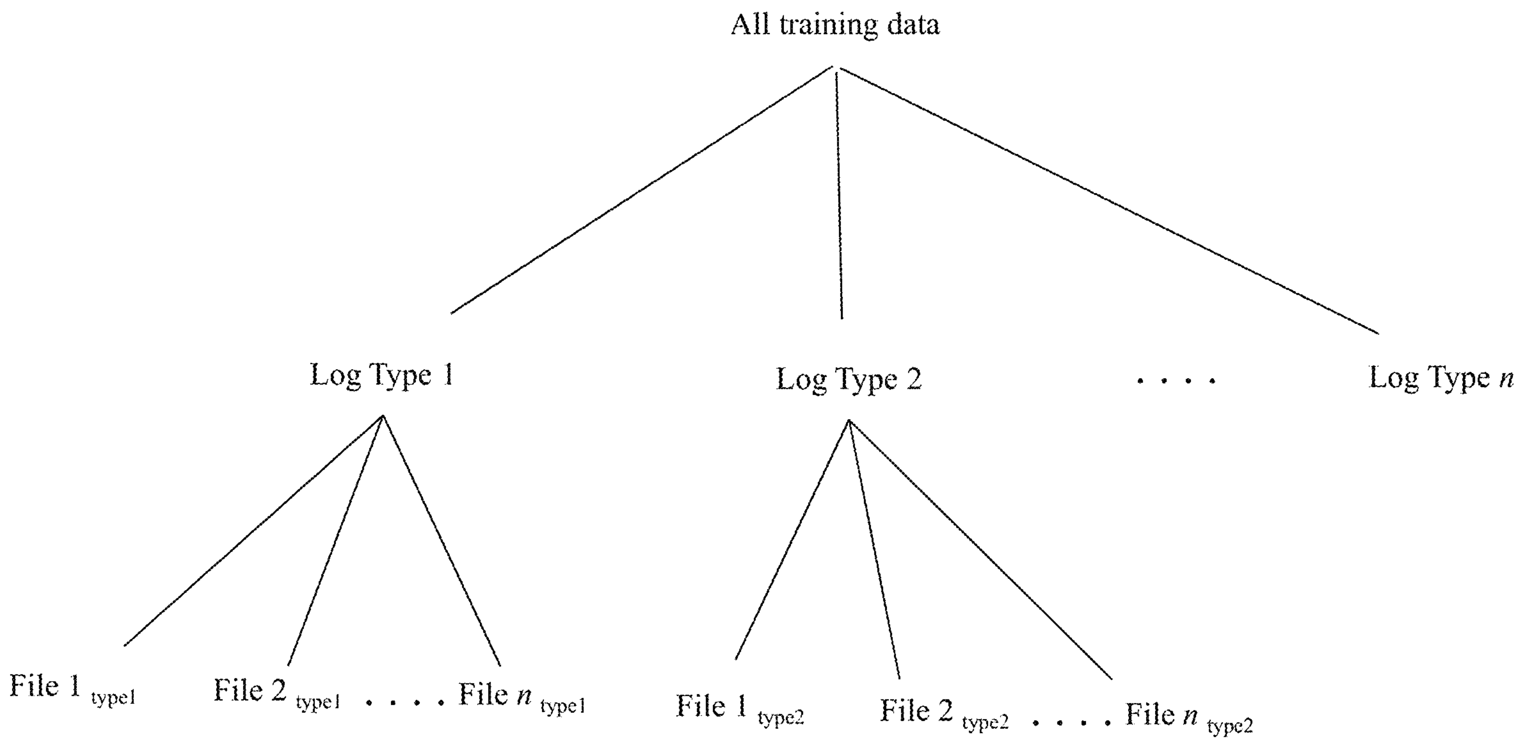

All training data
Log Type 1
Log Type 2
. . . .
Log Type *n*
File 1 type1
File 2 type1
. . . .
File *n* type1
File 1 type2
File 2 type2
. . . .
File *n* type2

Fig. 18

Log 1

```
Name=Bob     Date= May 1  URL=www.xyz.com/abcdefghijk/lmnopq
```

Log 2

```
Name=Joe     Date= April 5  URL=www.123.com/4567893409344/dfjgoms
```

Log 3

```
Name=Sam     Date= June 25  URL=www.abc.com/rtiosprmskfl/eroskuf
```

Fig. 19-1

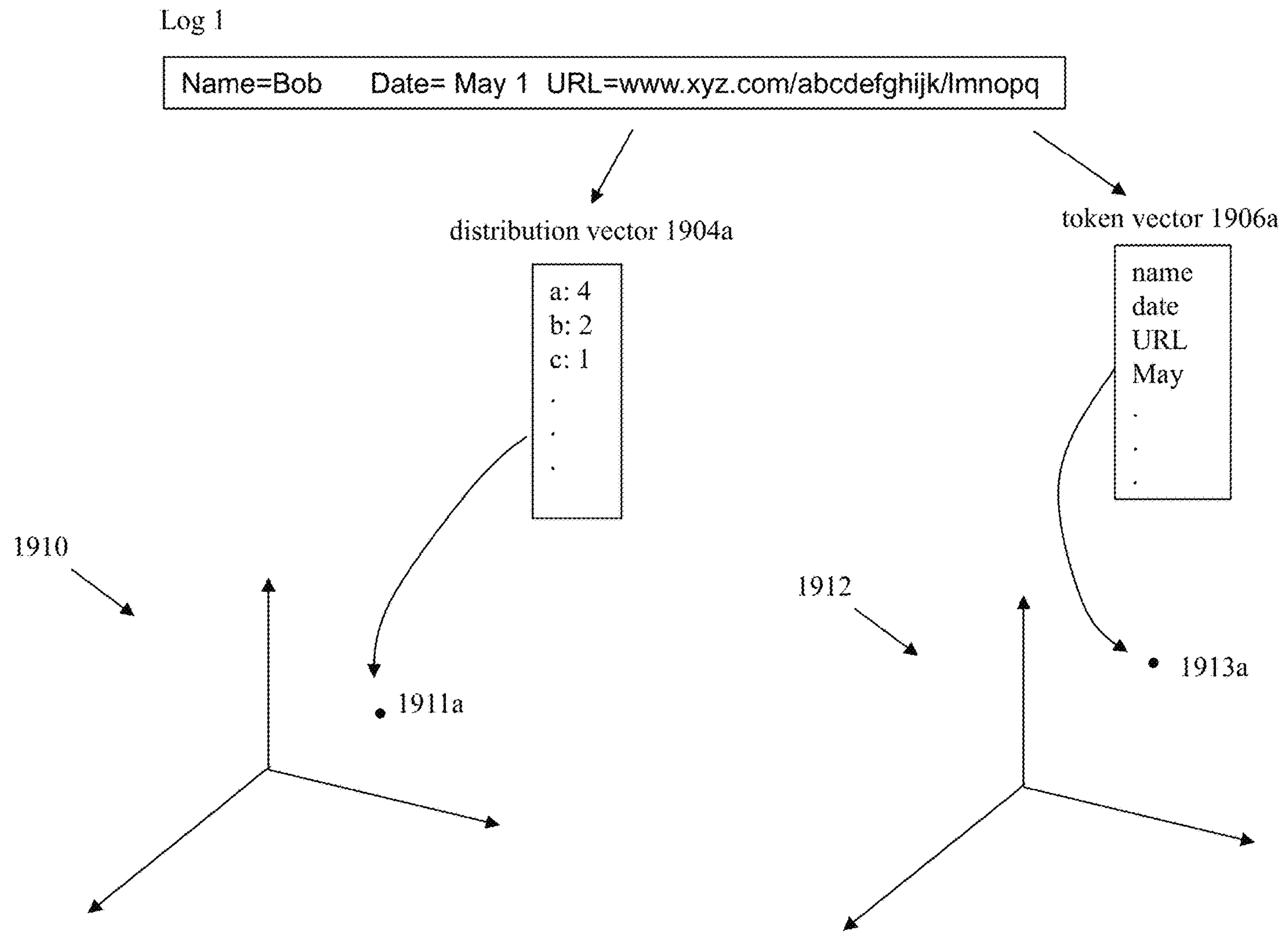

Log 1
Name=Bob Date= May 1 URL=www.xyz.com/abcdefghijk/lmnopq
distribution vector 1904a
a: 4
b: 2
c: 1
. . .
token vector 1906a
name
date
URL
May
. . .
1910
1911a
1912
1913a

Fig. 19-2

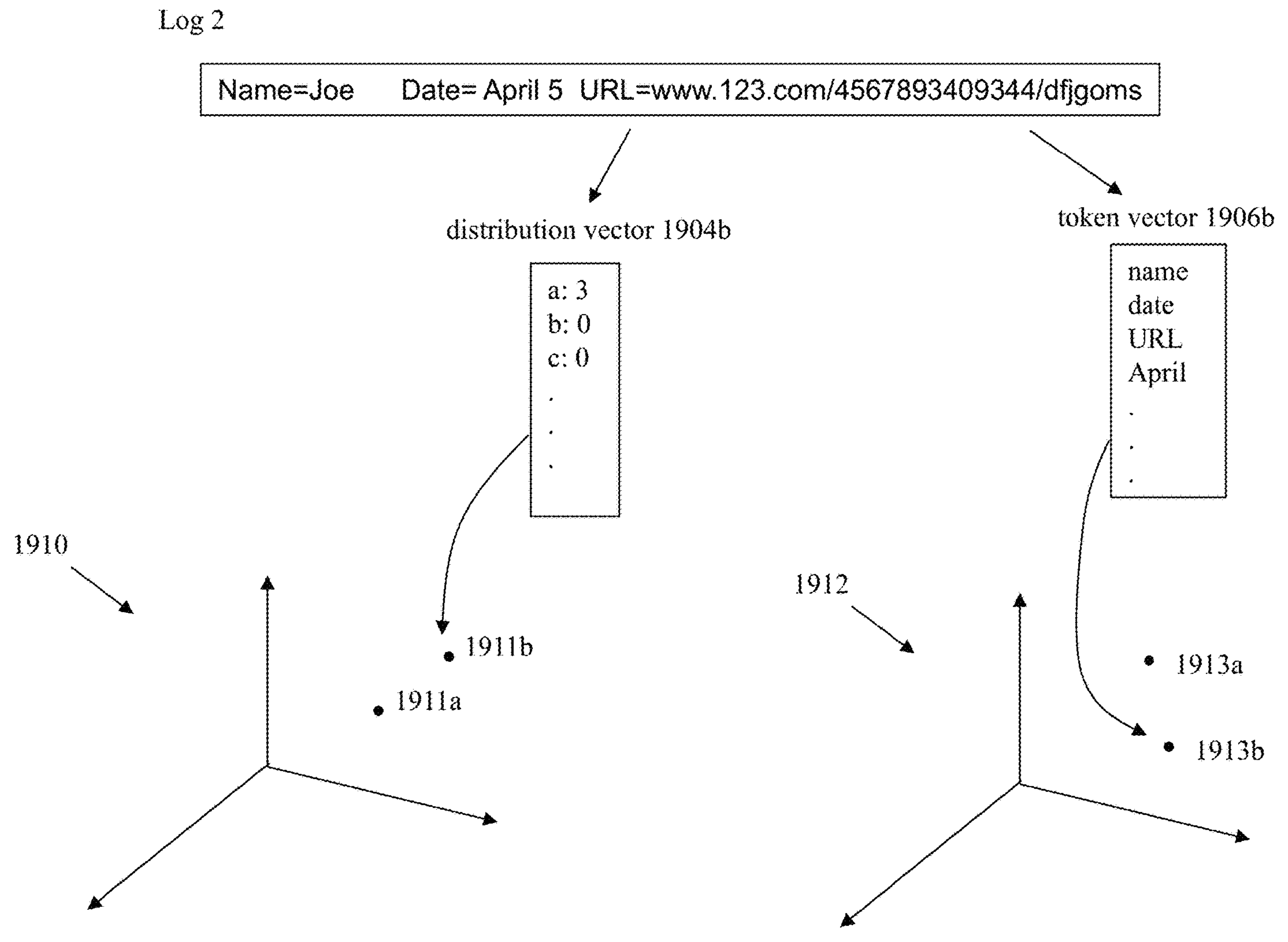
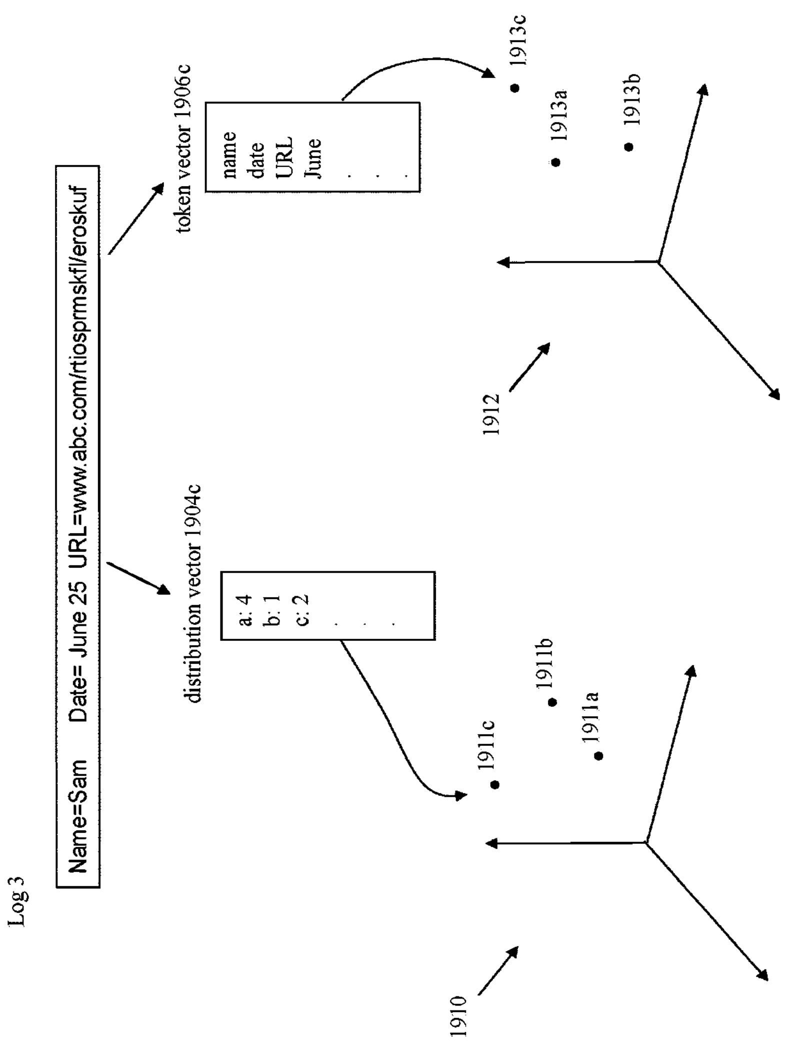

Log 2
Name=Joe Date= April 5 URL=www.123.com/4567893409344/dfjgoms
distribution vector 1904b
a: 3
b: 0
c: 0
.
.
.
token vector 1906b
name
date
URL
April
.
.
.
1910
1911b
1911a
1912
1913a
1913b

Fig. 19-3



```
Name=Sam    Date= June 25  URL=www.abc.com/rtiosprmskfl/eroskuf
```

Fig. 19-4

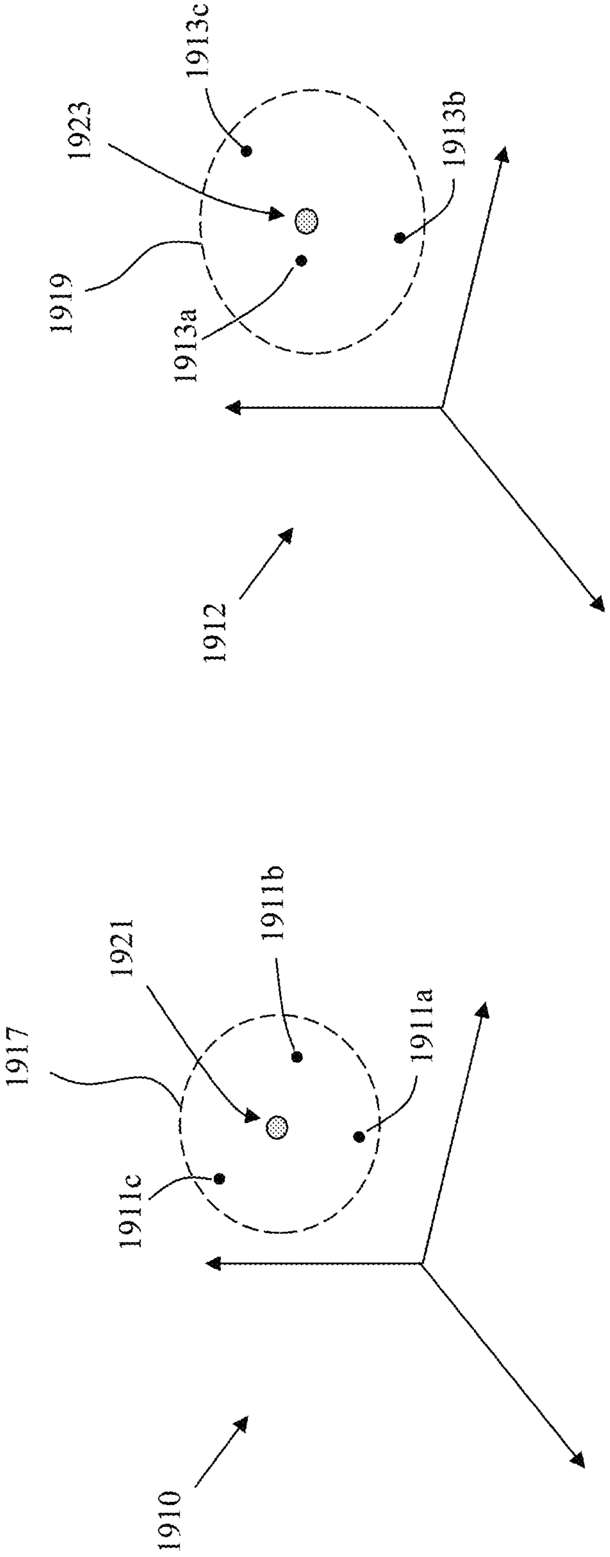

1910
1917
1921
1911b
1911a
1911c
1912
1919
1923
1913c
1913b
1913a

Fig. 19-5

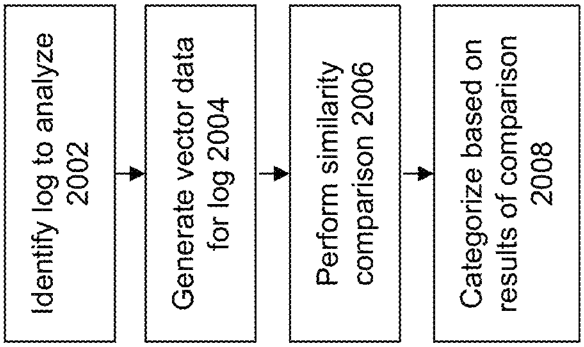

Identify log to analyze 2002
Generate vector data for log 2004
Perform similarity comparison 2006
Categorize based on results of comparison 2008

Fig. 20

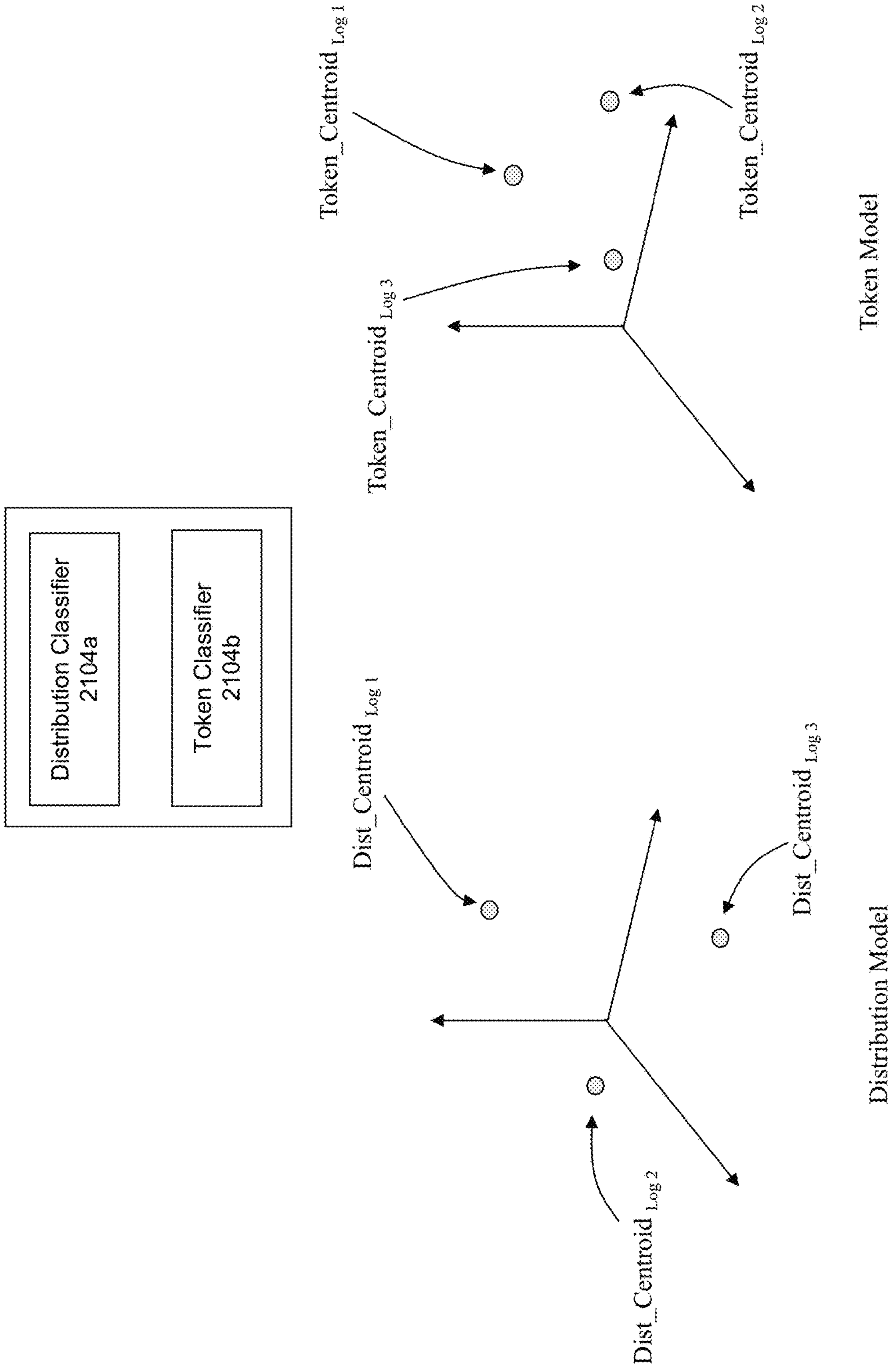

Distribution Classifier 2104a
Token Classifier 2104b
Token_Centroid Log 1
Token_Centroid Log 2
Token_Centroid Log 3
Token Model
Dist_Centroid Log 1
Dist_Centroid Log 3
Dist_Centroid Log 2
Distribution Model

Fig. 21-1

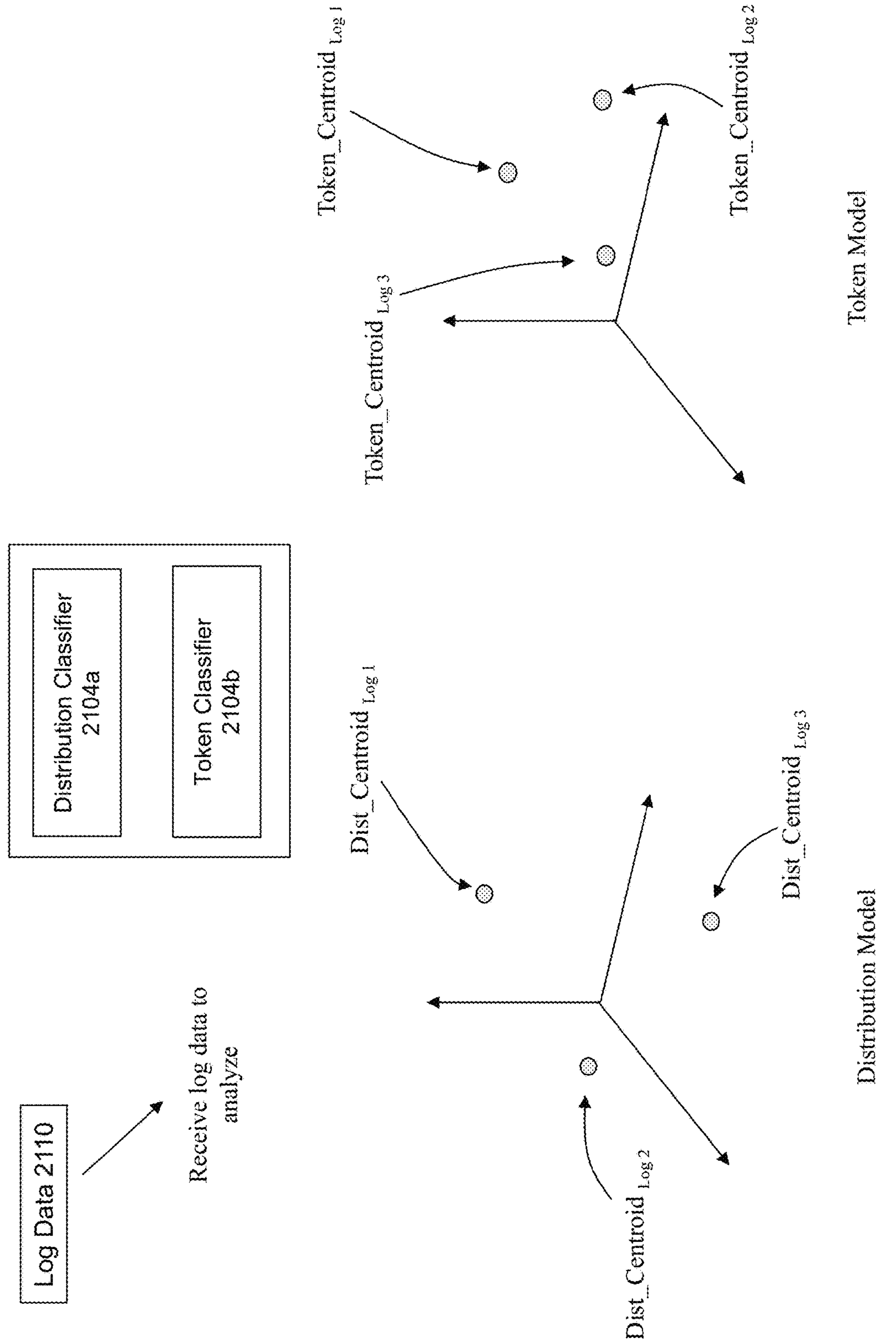

Log Data 2110
Receive log data to analyze
Distribution Classifier 2104a
Token Classifier 2104b
Dist_Centroid Log 1
Dist_Centroid Log 2
Dist_Centroid Log 3
Distribution Model
Token_Centroid Log 1
Token_Centroid Log 2
Token_Centroid Log 3
Token Model

Fig. 21-2

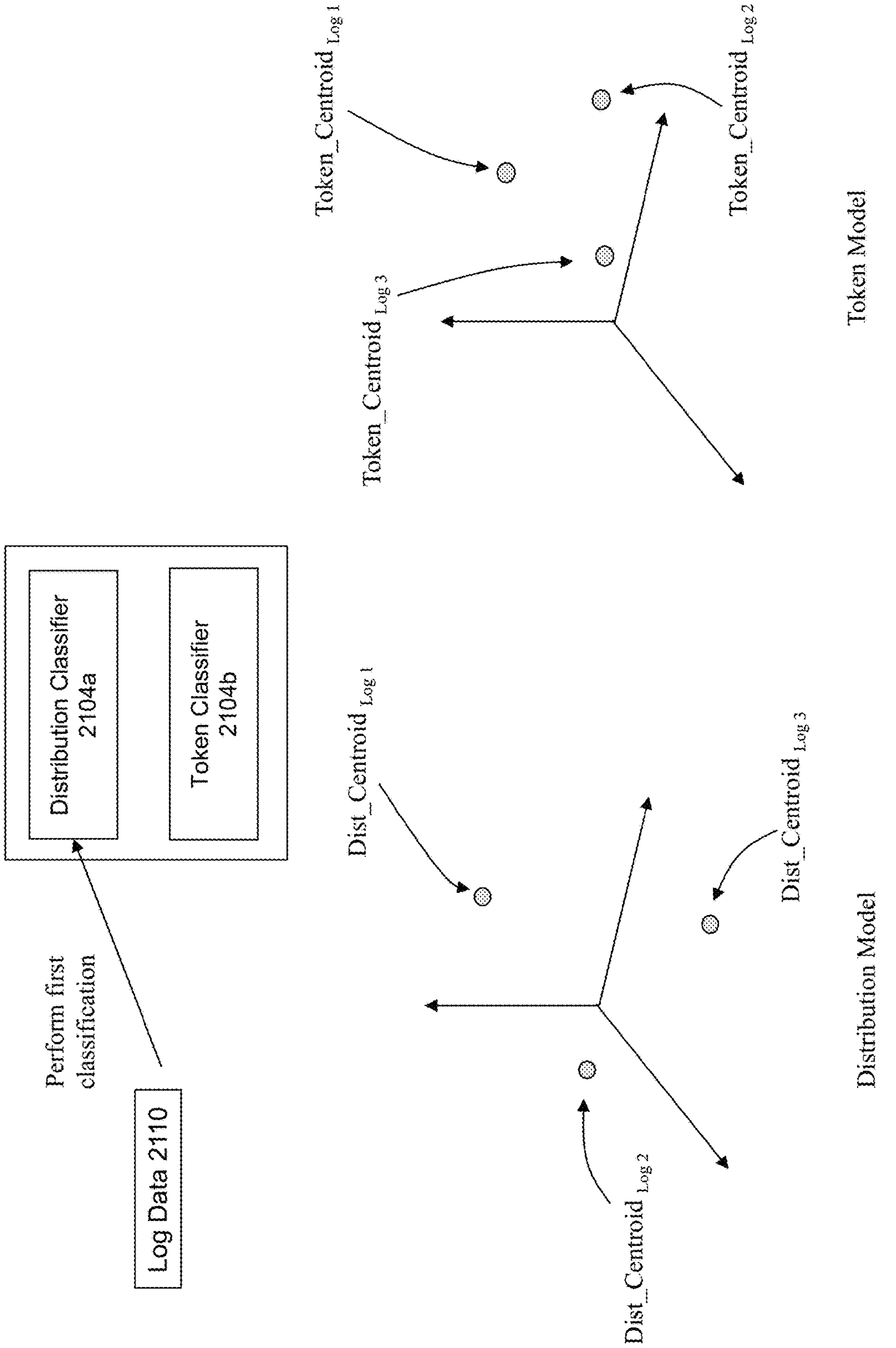

Log Data 2110
Perform first classification
Distribution Classifier 2104a
Token Classifier 2104b
Dist_Centroid$_{Log\ 2}$
Dist_Centroid$_{Log\ 1}$
Dist_Centroid$_{Log\ 3}$
Distribution Model
Token_Centroid$_{Log\ 3}$
Token_Centroid$_{Log\ 1}$
Token_Centroid$_{Log\ 2}$
Token Model

Fig. 21-3

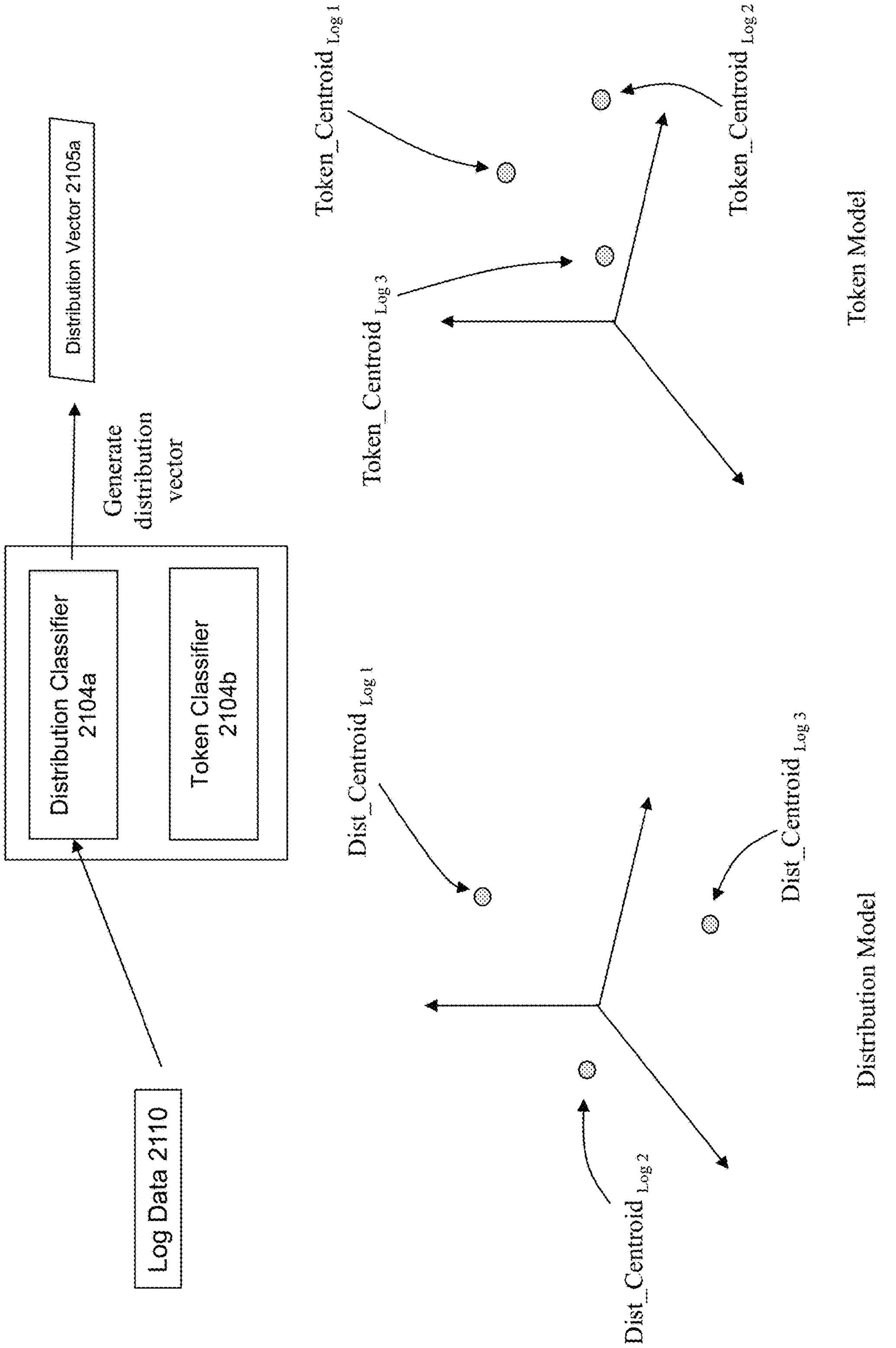

Log Data 2110
Distribution Classifier 2104a
Token Classifier 2104b
Generate distribution vector
Distribution Vector 2105a
Dist_Centroid $_{\text{Log 1}}$
Dist_Centroid $_{\text{Log 2}}$
Dist_Centroid $_{\text{Log 3}}$
Distribution Model
Token_Centroid $_{\text{Log 1}}$
Token_Centroid $_{\text{Log 2}}$
Token_Centroid $_{\text{Log 3}}$
Token Model

Fig. 21-4

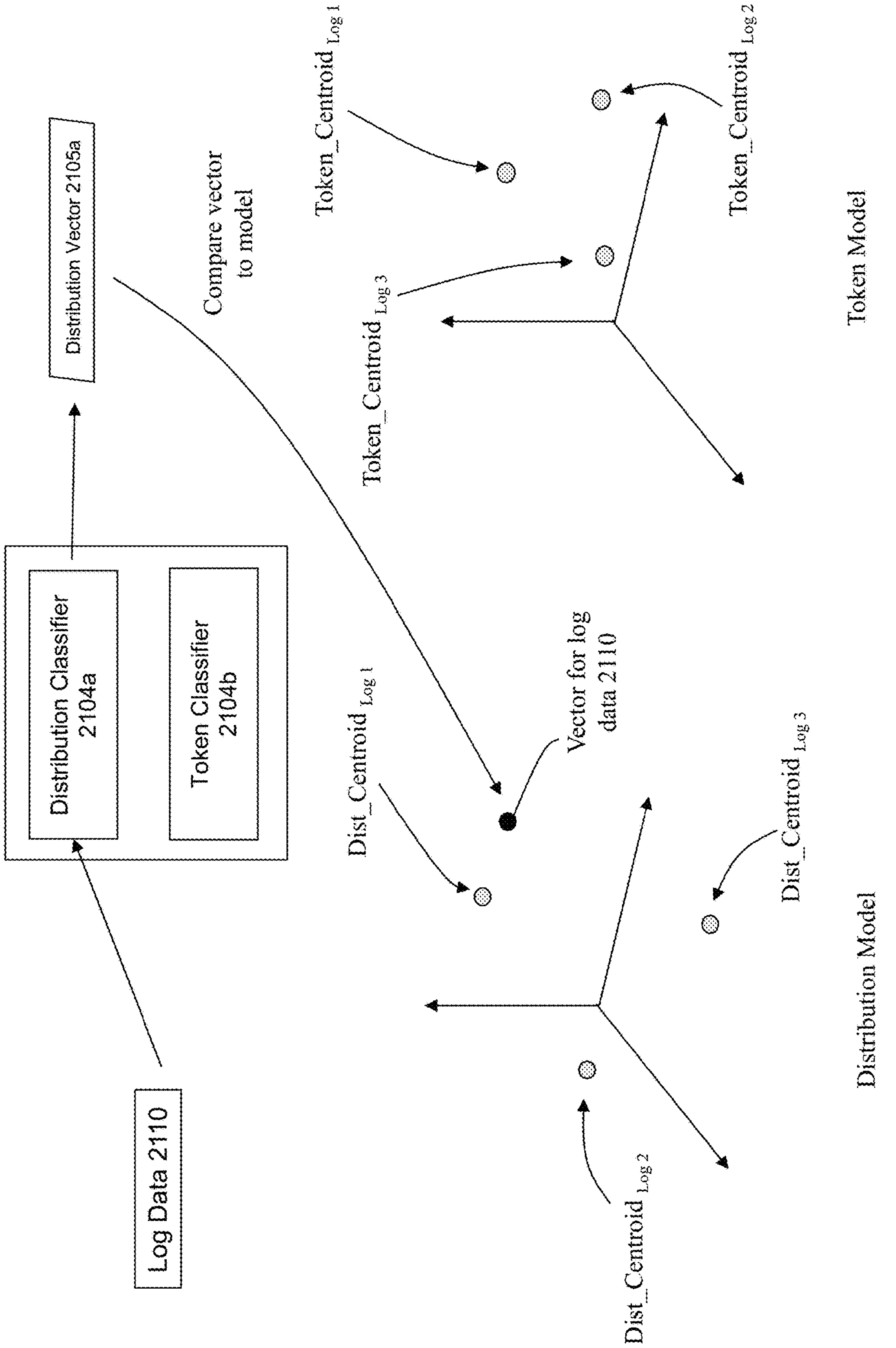

Log Data 2110
Distribution Classifier 2104a
Token Classifier 2104b
Distribution Vector 2105a
Compare vector to model
Vector for log data 2110
Token_Centroid $_{Log\ 1}$
Token_Centroid $_{Log\ 2}$
Token_Centroid $_{Log\ 3}$
Token Model
Dist_Centroid $_{Log\ 1}$
Dist_Centroid $_{Log\ 2}$
Dist_Centroid $_{Log\ 3}$
Distribution Model

Fig. 21-5

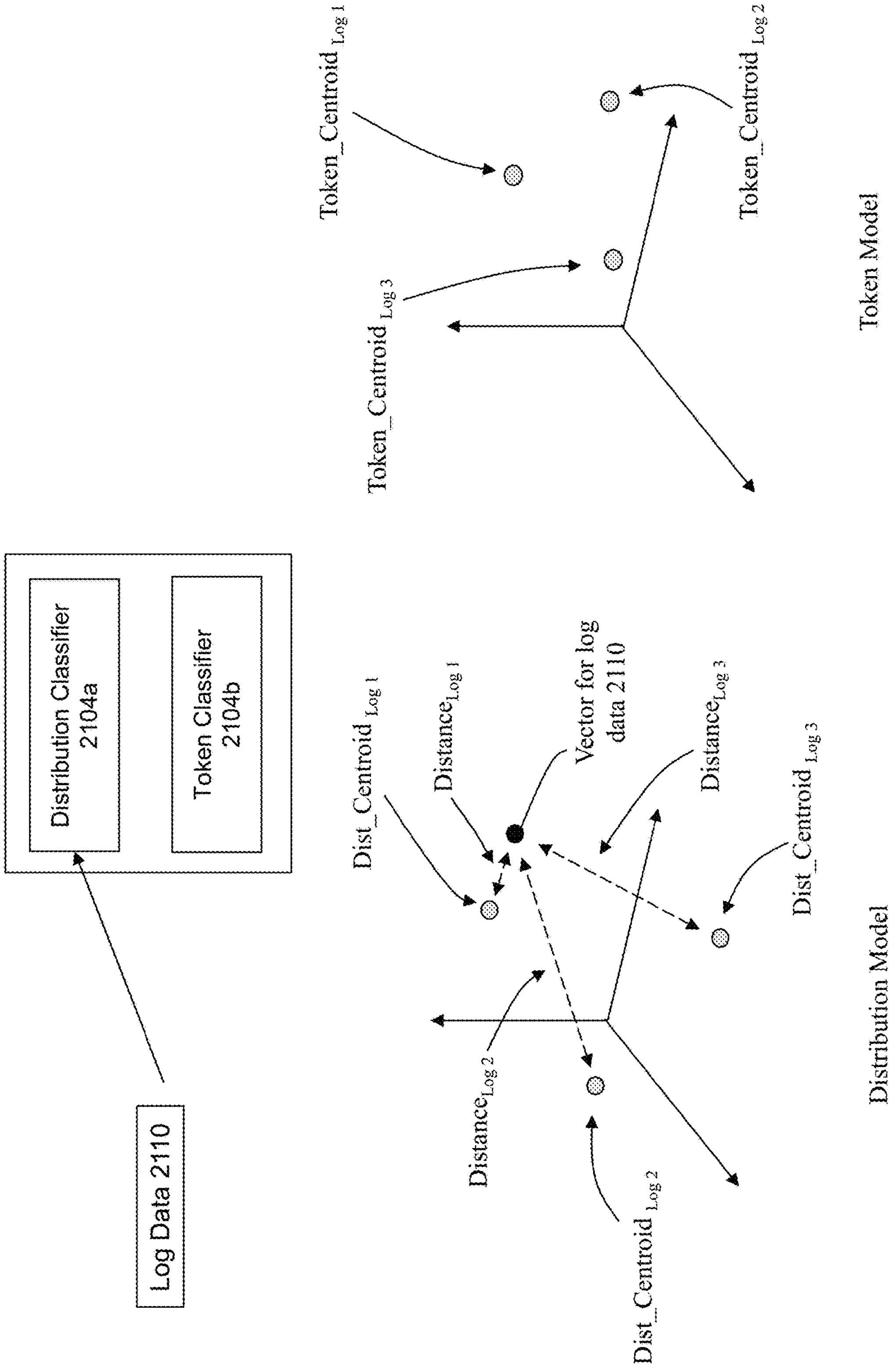

Log Data 2110
Distribution Classifier 2104a
Token Classifier 2104b
$\text{Dist\_Centroid}_{\text{Log 1}}$
$\text{Distance}_{\text{Log 1}}$
Vector for log data 2110
$\text{Distance}_{\text{Log 3}}$
$\text{Dist\_Centroid}_{\text{Log 3}}$
$\text{Distance}_{\text{Log 2}}$
$\text{Dist\_Centroid}_{\text{Log 2}}$
Distribution Model
$\text{Token\_Centroid}_{\text{Log 1}}$
$\text{Token\_Centroid}_{\text{Log 2}}$
$\text{Token\_Centroid}_{\text{Log 3}}$
Token Model

Fig. 21-6

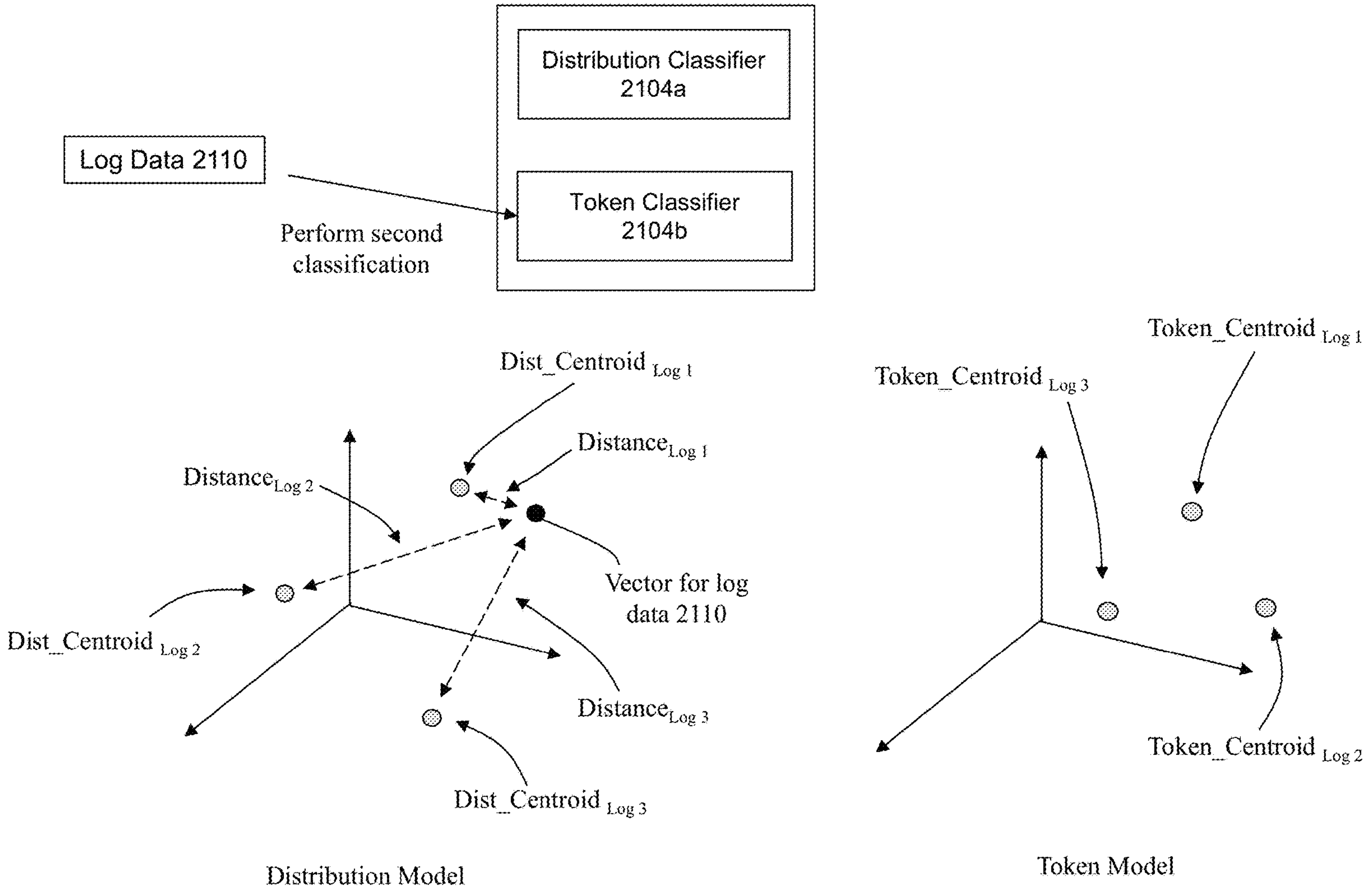

Log Data 2110
Perform second classification
Distribution Classifier 2104a
Token Classifier 2104b
Dist_Centroid $_{Log\ 1}$
Distance$_{Log\ 1}$
Distance$_{Log\ 2}$
Vector for log data 2110
Dist_Centroid $_{Log\ 2}$
Distance$_{Log\ 3}$
Dist_Centroid $_{Log\ 3}$
Distribution Model
Token_Centroid $_{Log\ 1}$
Token_Centroid $_{Log\ 3}$
Token_Centroid $_{Log\ 2}$
Token Model

Fig. 21-7

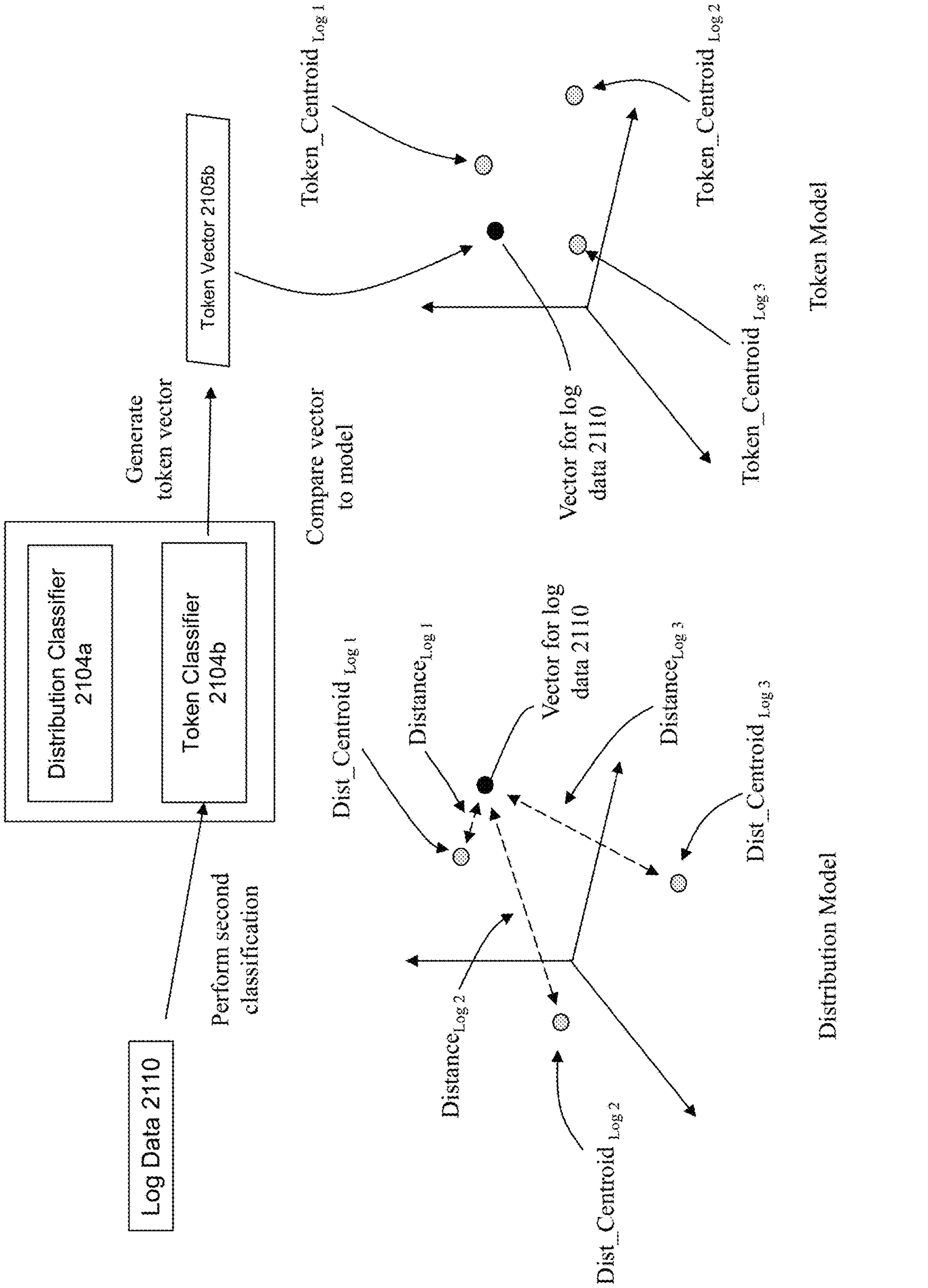

Log Data 2110
Perform second classification
Distribution Classifier 2104a
Token Classifier 2104b
Generate token vector
Token Vector 2105b
Compare vector to model
Token_Centroid $_{Log\ 1}$
Token_Centroid $_{Log\ 2}$
Token_Centroid $_{Log\ 3}$
Vector for log data 2110
Token Model
Dist_Centroid $_{Log\ 1}$
Distance$_{Log\ 1}$
Vector for log data 2110
Distance$_{Log\ 3}$
Dist_Centroid $_{Log\ 3}$
Distance$_{Log\ 2}$
Dist_Centroid $_{Log\ 2}$
Distribution Model

Fig. 21-8

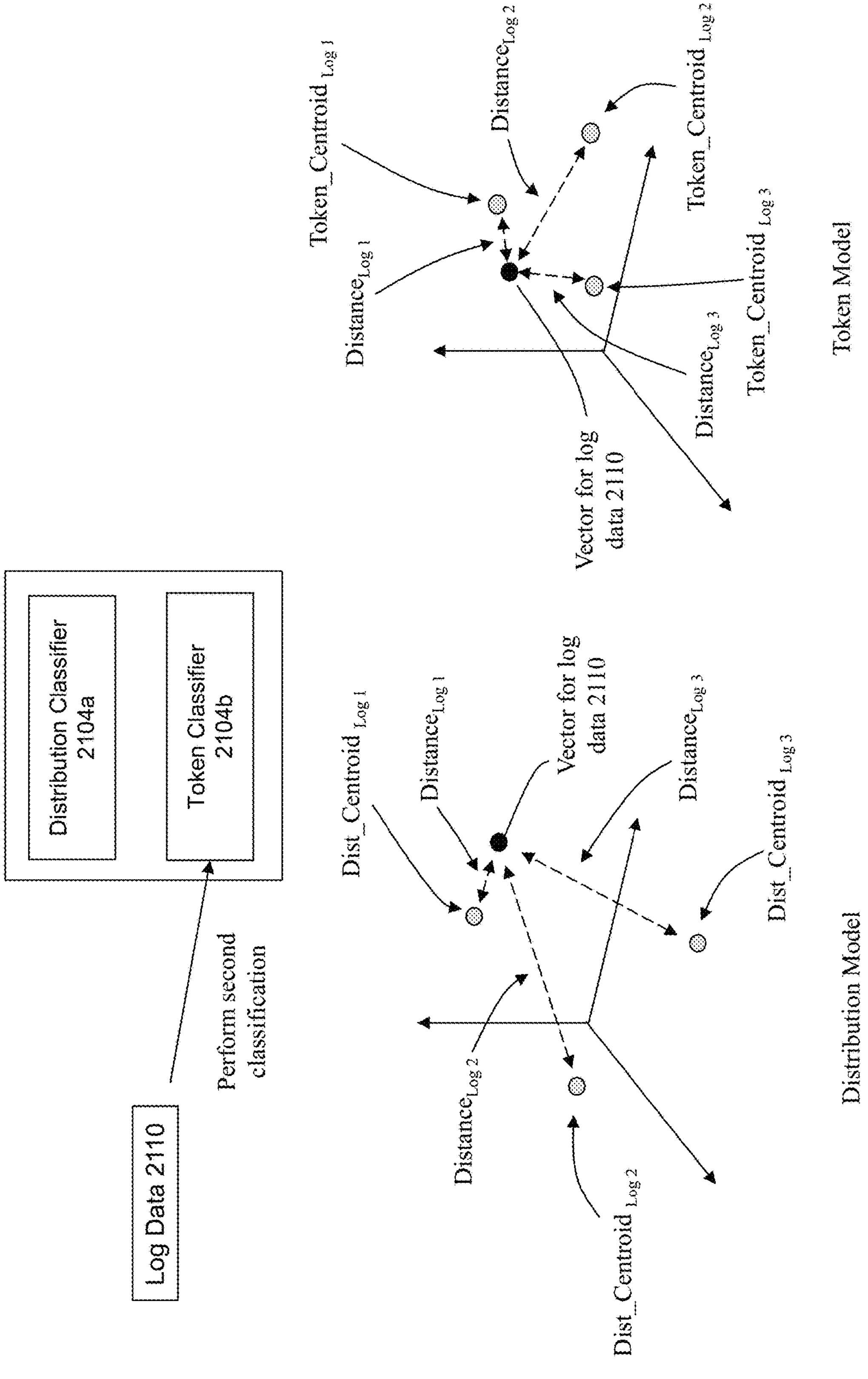
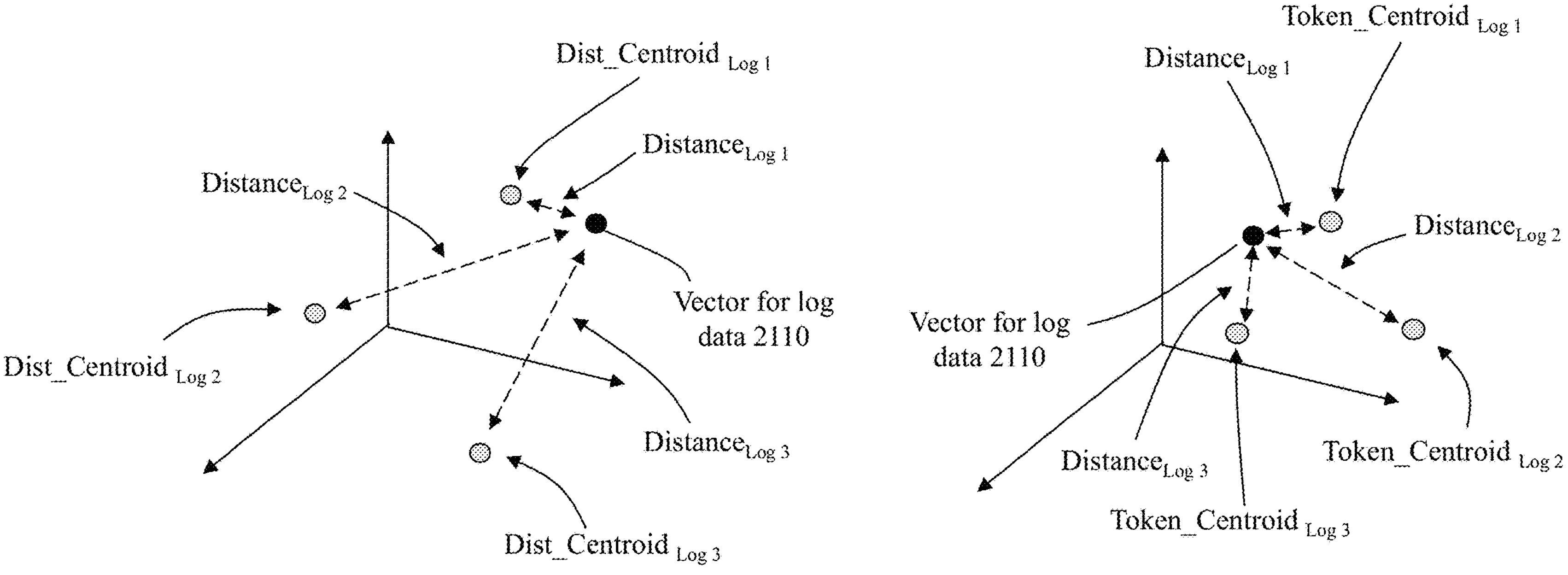

Log Data 2110
Perform second classification
Distribution Classifier 2104a
Token Classifier 2104b
Dist_Centroid $_{Log 1}$
Distance$_{Log 1}$
Vector for log data 2110
Distance$_{Log 3}$
Dist_Centroid $_{Log 3}$
Distance$_{Log 2}$
Dist_Centroid $_{Log 2}$
Distribution Model
Token_Centroid $_{Log 1}$
Distance$_{Log 1}$
Distance$_{Log 2}$
Token_Centroid $_{Log 2}$
Token_Centroid $_{Log 3}$
Distance$_{Log 3}$
Vector for log data 2110
Token Model

Fig. 21-9

$$\text{Probability Log 1} = \text{Weight}_{\text{Dist\_Model}} * \text{Prob\_Dist}(\text{Distance}_{\text{Log 1}}) + \text{Weight}_{\text{Token\_Model}} * \text{Prob\_Token}(\text{Distance}_{\text{Log 1}})$$

$$\text{Probability Log 2} = \text{Weight}_{\text{Dist\_Model}} * \text{Prob\_Dist}(\text{Distance}_{\text{Log 2}}) + \text{Weight}_{\text{Token\_Model}} * \text{Prob\_Token}(\text{Distance}_{\text{Log 2}})$$

$$\text{Probability Log 3} = \text{Weight}_{\text{Dist\_Model}} * \text{Prob\_Dist}(\text{Distance}_{\text{Log 3}}) + \text{Weight}_{\text{Token\_Model}} * \text{Prob\_Token}(\text{Distance}_{\text{Log 3}})$$


Dist_Centroid $_{\text{Log 1}}$
Distance$_{\text{Log 1}}$
Distance$_{\text{Log 2}}$
Vector for log data 2110
Dist_Centroid $_{\text{Log 2}}$
Distance$_{\text{Log 3}}$
Dist_Centroid $_{\text{Log 3}}$
Token_Centroid $_{\text{Log 1}}$
Distance$_{\text{Log 1}}$
Distance$_{\text{Log 2}}$
Vector for log data 2110
Distance$_{\text{Log 3}}$
Token_Centroid $_{\text{Log 2}}$
Token_Centroid $_{\text{Log 3}}$

Distribution Model

Token Model

Fig. 21-10

Log Type	Probability
Log 1	80%
Log 2	10%
Log 3	20%

Fig. 21-11

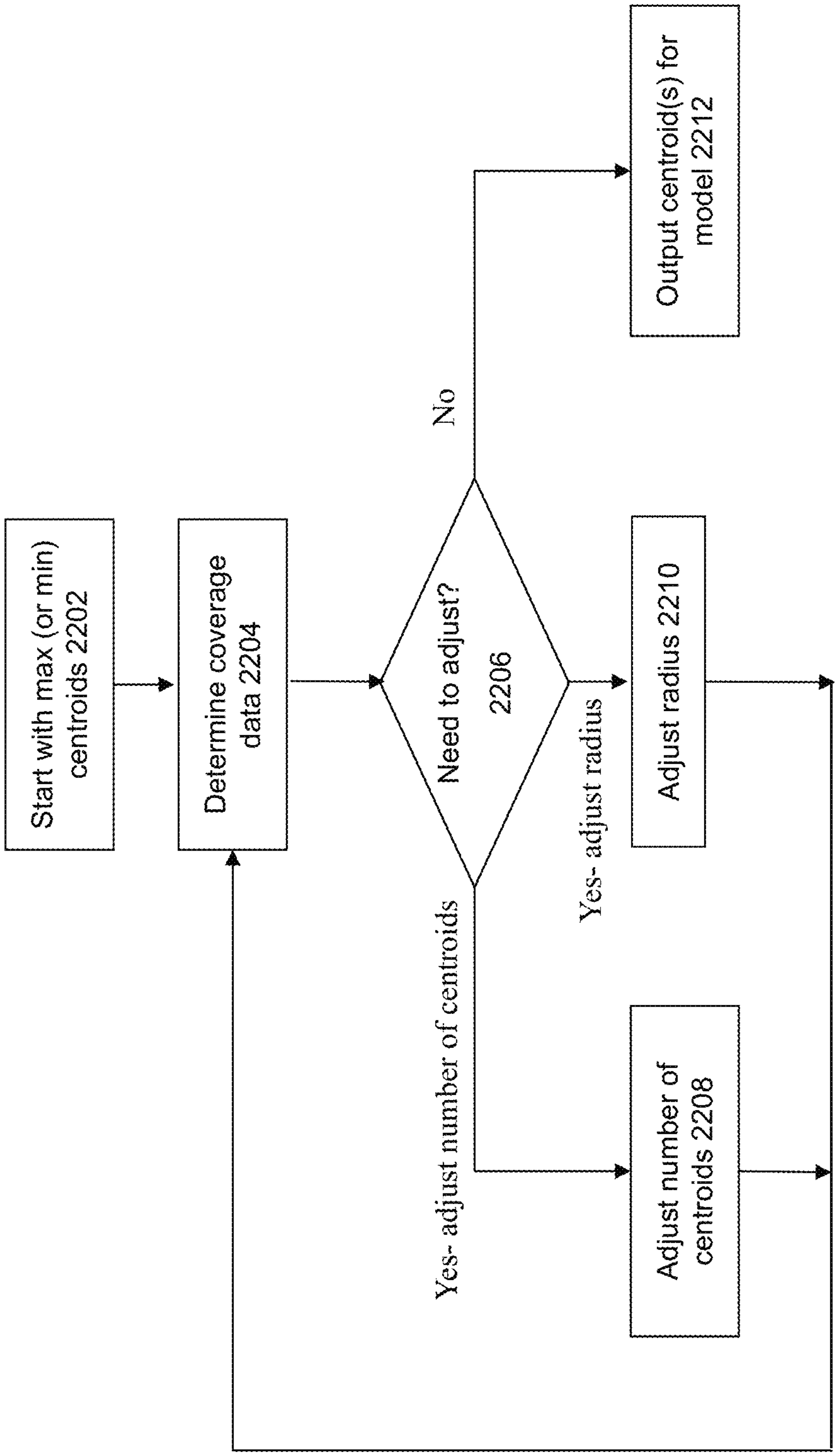

Start with max (or min) centroids 2202
Determine coverage data 2204
Need to adjust? 2206
No
Output centroid(s) for model 2212
Yes- adjust radius
Adjust radius 2210
Yes- adjust number of centroids
Adjust number of centroids 2208

Fig. 22

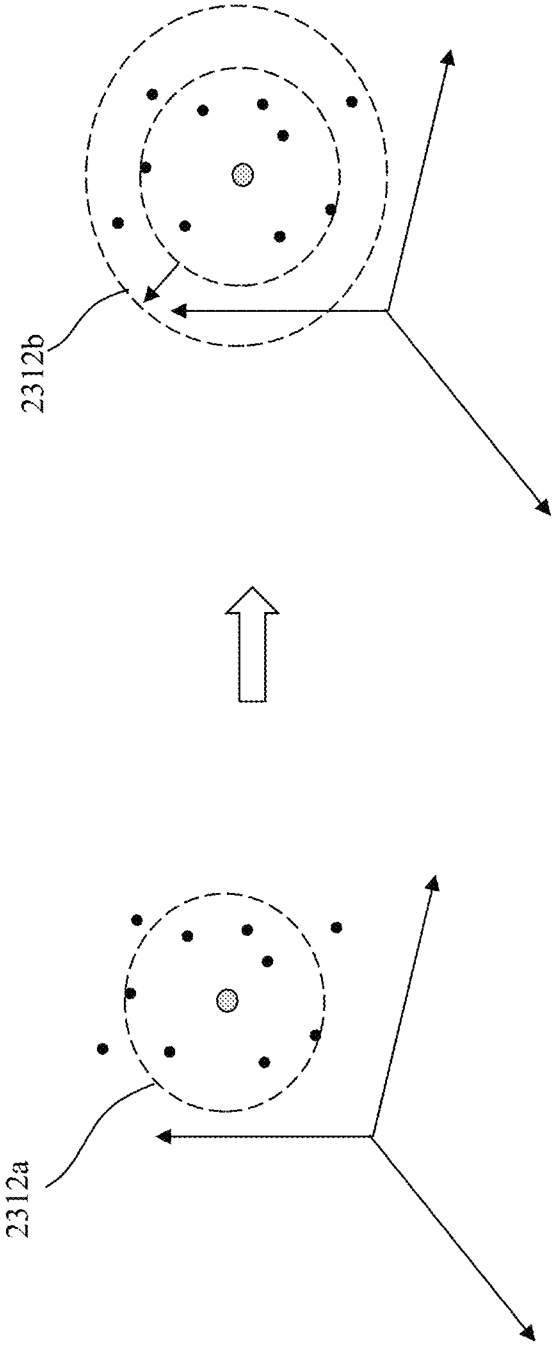


Fig. 23

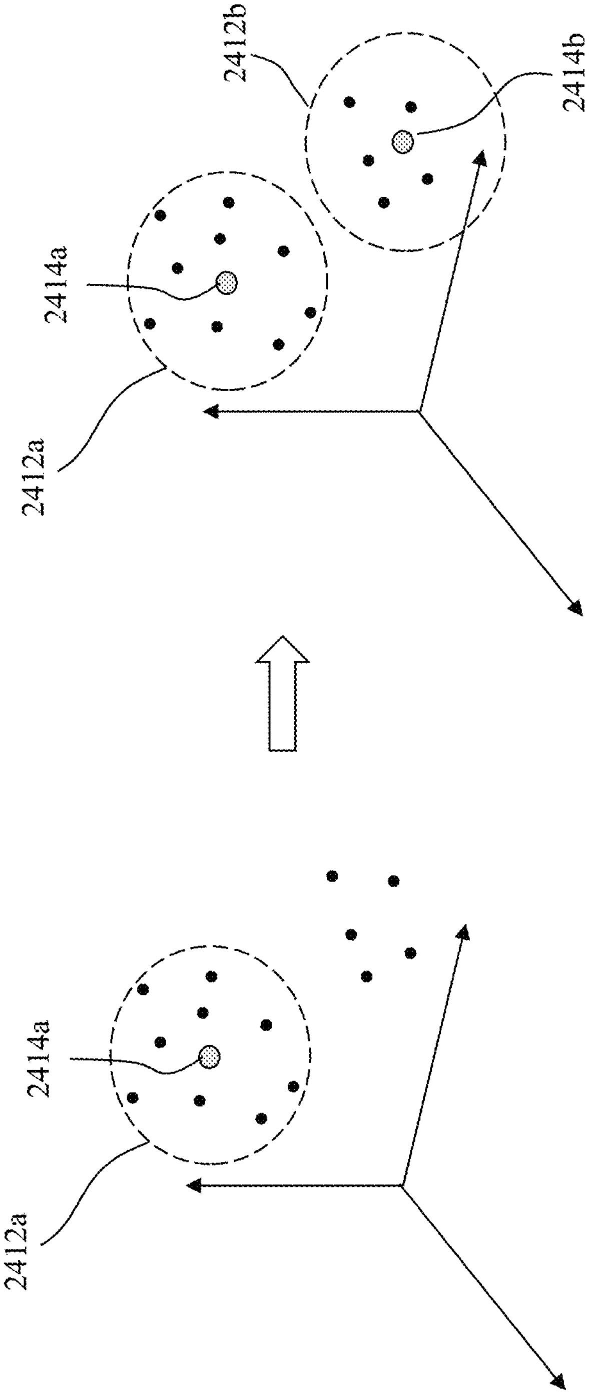


Fig. 24

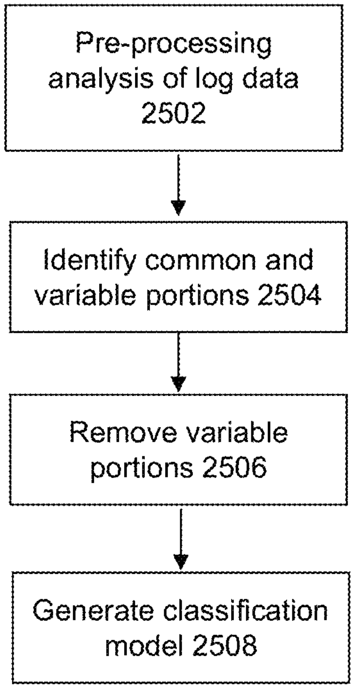
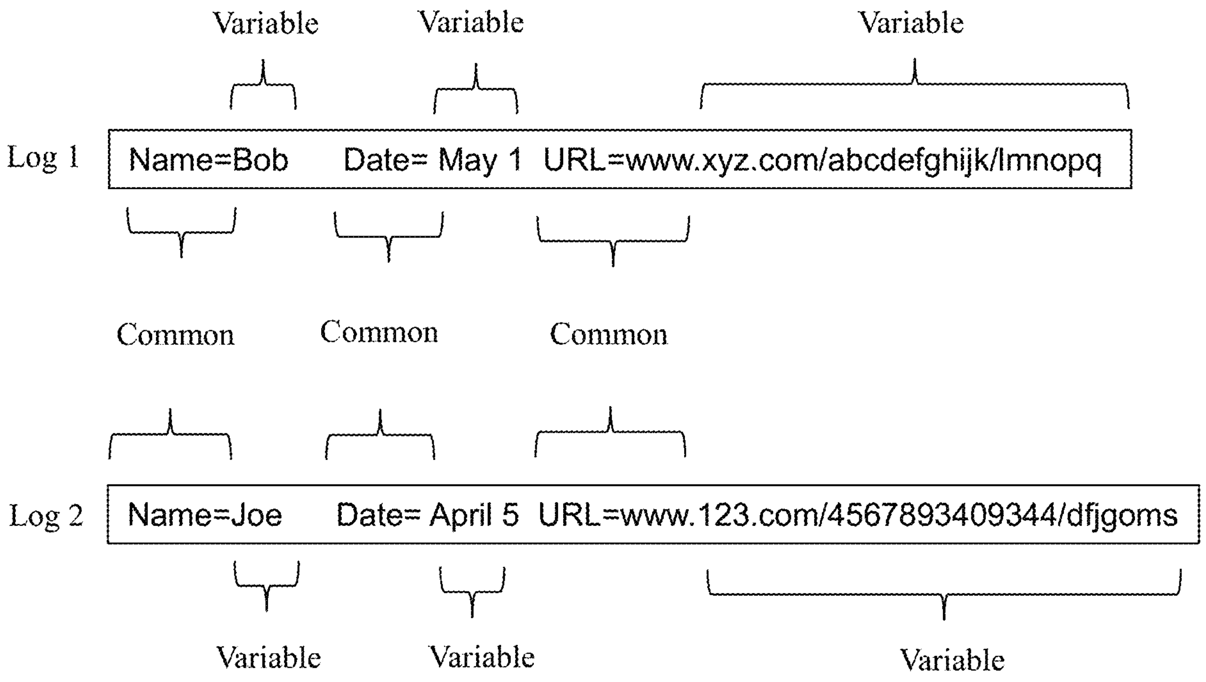

Pre-processing analysis of log data 2502
Identify common and variable portions 2504
Remove variable portions 2506
Generate classification model 2508

Fig. 25

```
Log 1   Name=Bob     Date= May 1  URL=www.xyz.com/abcdefghijk/lmnopq

Log 2   Name=Joe     Date= April 5  URL=www.123.com/4567893409344/dfjgoms
```

Fig. 26-1



```
Log 1  Name=Bob     Date= May 1  URL=www.xyz.com/abcdefghijk/lmnopq
Log 2  Name=Joe     Date= April 5  URL=www.123.com/4567893409344/dfjgoms
```

Fig. 26-2

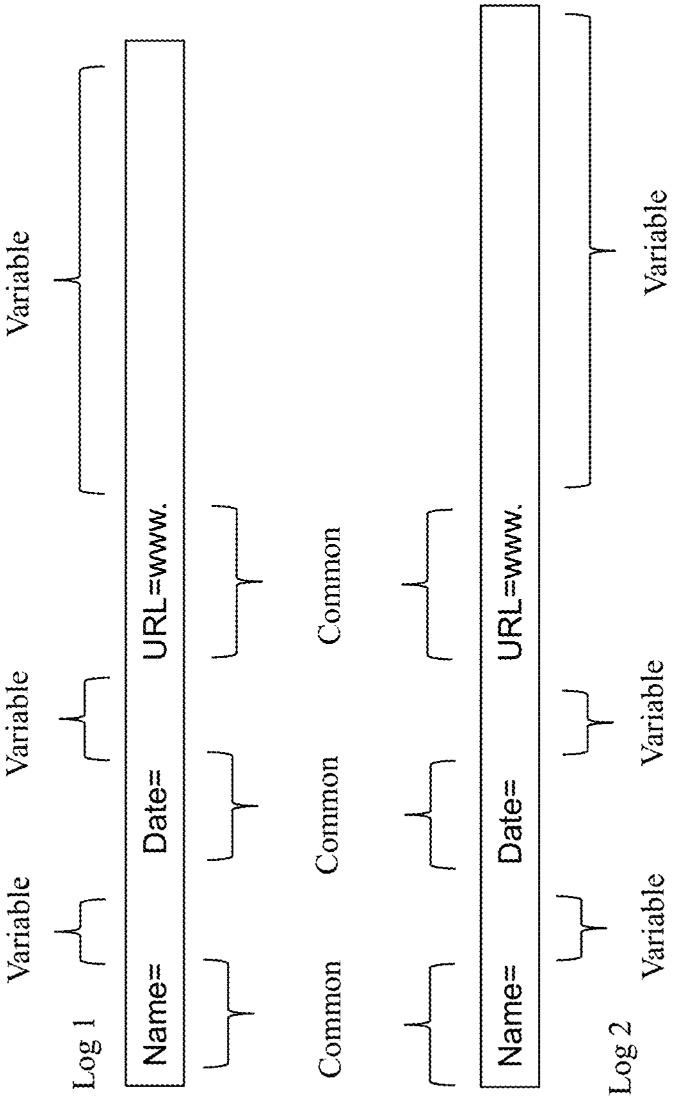

Log 1
Variable
Variable
Variable
Name=
Date=
URL=www.
Common
Common
Common
Name=
Date=
URL=www.
Log 2
Variable
Variable
Variable

Fig. 26-3

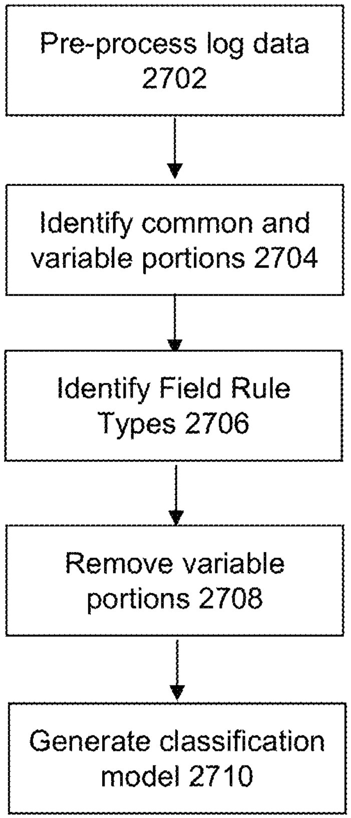
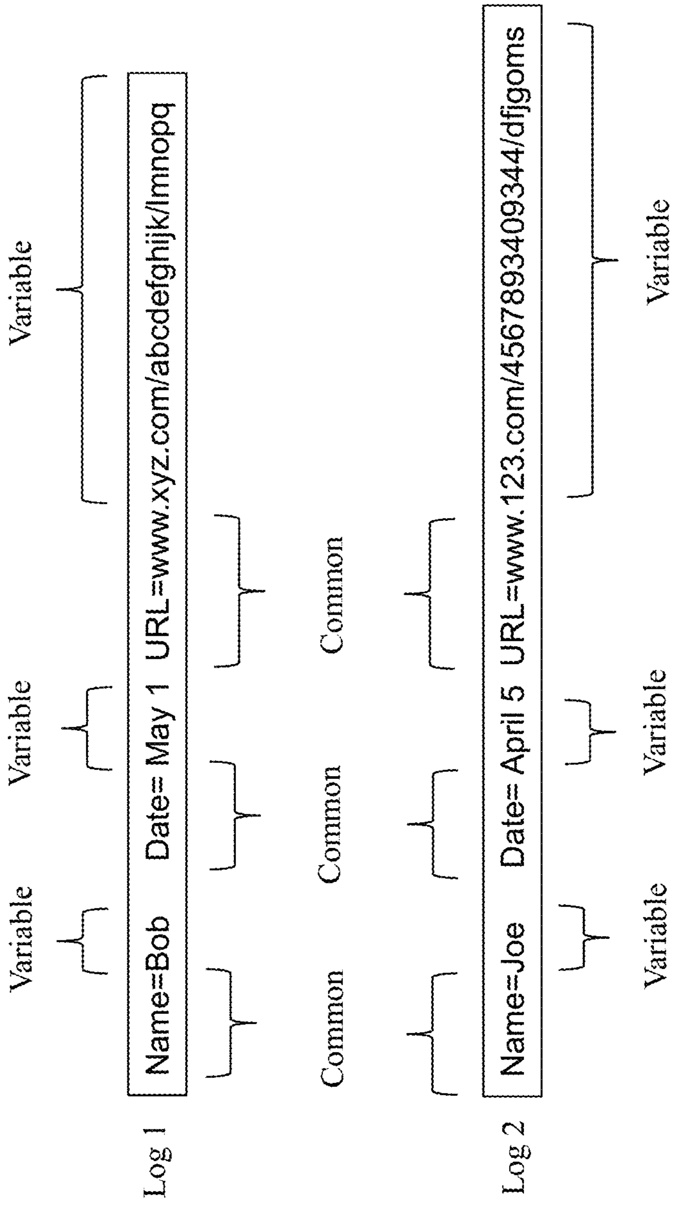

Pre-process log data 2702
Identify common and variable portions 2704
Identify Field Rule Types 2706
Remove variable portions 2708
Generate classification model 2710

Fig. 27

```
Log 1   Name=Bob     Date= May 1  URL=www.xyz.com/abcdefghijk/lmnopq

Log 2   Name=Joe     Date= April 5  URL=www.123.com/4567893409344/dfjgoms
```

Fig. 28-1



```
Log 1   Name=Bob   Date= May 1  URL=www.xyz.com/abcdefghijk/lmnopq

Log 2   Name=Joe   Date= April 5  URL=www.123.com/4567893409344/dfjgoms
```

Fig. 28-2

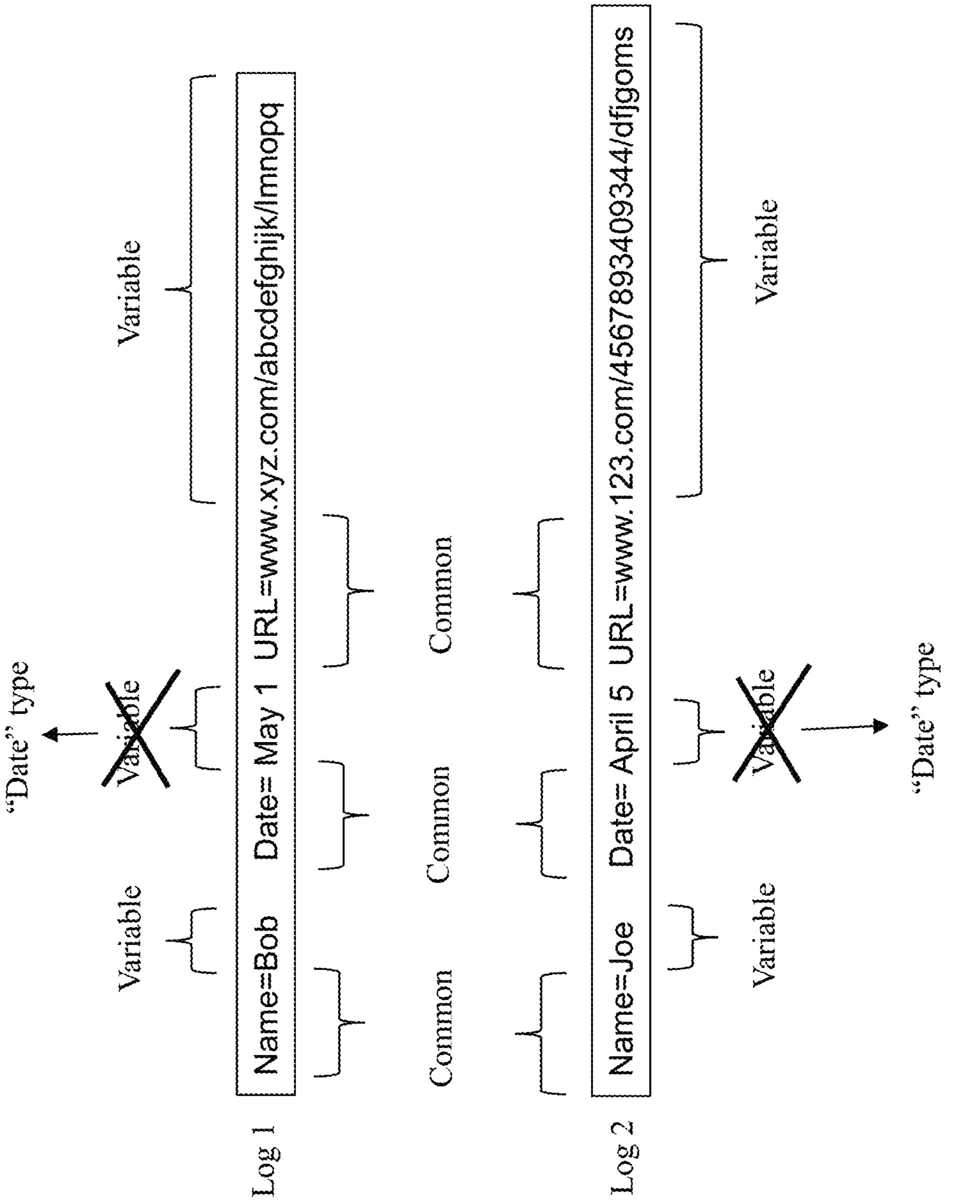

Log 1
Name=Bob Date= May 1 URL=www.xyz.com/abcdefghijk/lmnopq
Common
Variable
Common
~~Variable~~
"Date" type
Common
Variable
Log 2
Name=Joe Date= April 5 URL=www.123.com/4567893409344/dfjgoms
Common
Variable
Common
~~Variable~~
"Date" type
Common
Variable

Fig. 28-3

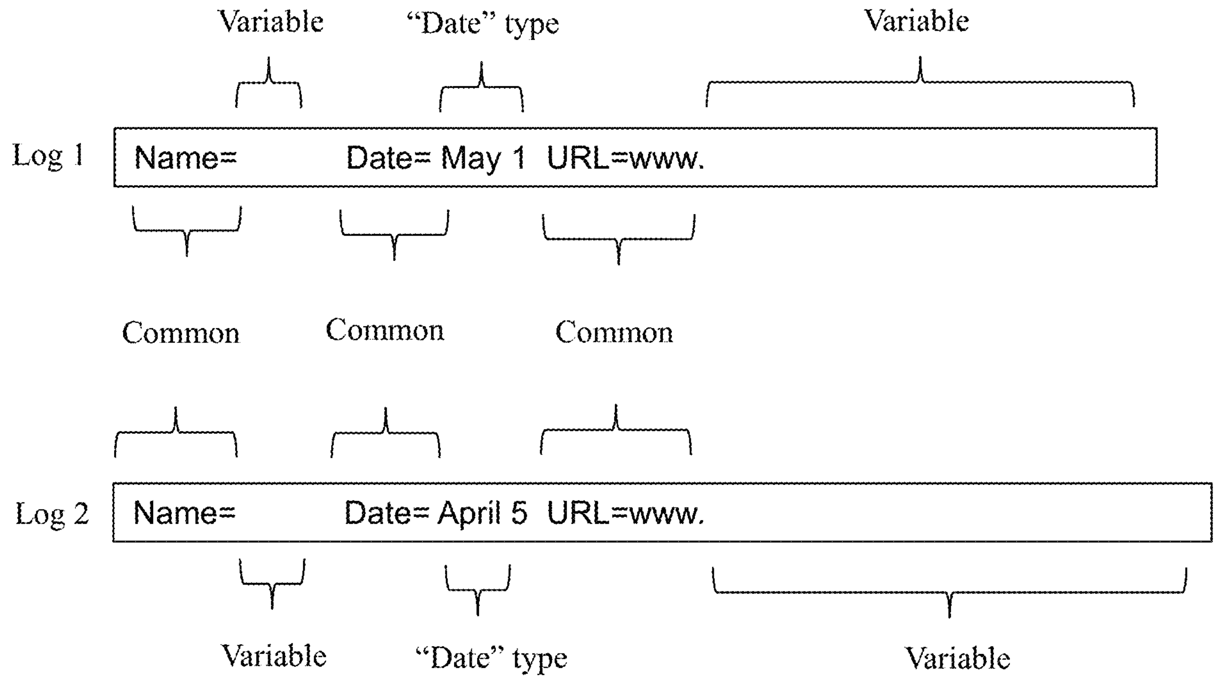

Variable
"Date" type
Variable
Log 1
Name=
Date= May 1
URL=www.
Common
Common
Common
Log 2
Name=
Date= April 5
URL=www.
Variable
"Date" type
Variable

Fig. 28-4

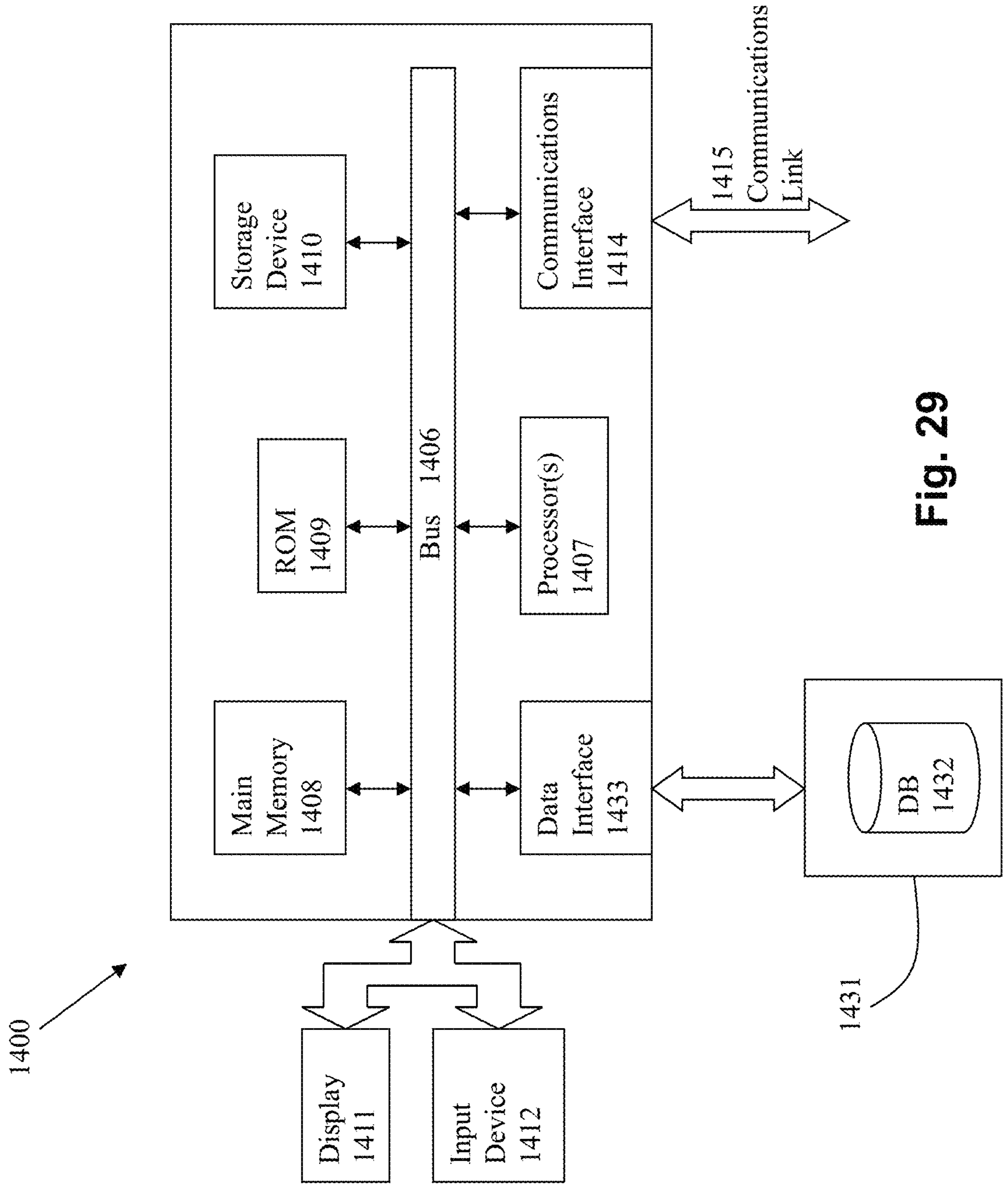

1400
Main Memory 1408
ROM 1409
Storage Device 1410
Bus 1406
Data Interface 1433
Processor(s) 1407
Communications Interface 1414
1415 Communications Link
DB 1432
1431
Display 1411
Input Device 1412

Fig. 29

METHOD AND SYSTEM FOR IMPLEMENTING MACHINE LEARNING CLASSIFICATIONS

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application claims the benefit of priority to U.S. Provisional Application No. 62/142,987, filed on Apr. 3, 2015, which is hereby incorporated by reference in its entirety. The present application is related to (a) U.S. Ser. No. 15/088,943, entitled "METHOD AND SYSTEM FOR IMPLEMENTING TARGET MODEL CONFIGURATION METADATA FOR A LOG ANALYTICS SYSTEM", (b) U.S. Ser. No. 15/089,005, entitled "METHOD AND SYSTEM FOR PARAMETERIZING LOG FILE LOCATION ASSIGNMENTS FOR A LOG ANALYTICS SYSTEM", (c) U.S. Ser. No. 15/089,049, entitled "METHOD AND SYSTEM FOR IMPLEMENTING AN OPERATING SYSTEM HOOK INA LOG ANALYTICS SYSTEM", (d) U.S. Ser. No. 15/089,129, entitled "METHOD AND SYSTEM FOR IMPLEMENTING COLLECTION-WISE PROCESSING IN A LOG ANALYTICS SYSTEM", (e) U.S. Ser. No. 15/089,180, entitled "METHOD AND SYSTEM FOR IMPLEMENTING A LOG PARSER IN A LOG ANALYTICS SYSTEM", all filed on even date herewith, and which are all hereby incorporated by reference in their entirety.

BACKGROUND AND SUMMARY

Many types of computing systems and applications generate vast amounts of data pertaining to or resulting from the operation of that computing system or application. These vast amounts of data are stored into collected locations, such as log files/records, which can then be reviewed at a later time period if there is a need to analyze the behavior or operation of the system or application.

Server administrators and application administrators can benefit by learning about and analyzing the contents of the system log records. However, it can be a very challenging task to collect and analyze these records. There are many reasons for these challenges.

One significant issue pertains to the fact that many modern organizations possess a very large number of computing systems, each having numerous applications that run on those computing systems. It can be very difficult in a large system to configure, collect, and analyze log records given the large number of disparate systems and applications that run on those computing devices. Furthermore, some of those applications may actually run on and across multiple computing systems, making the task of coordinating log configuration and collection even more problematic.

Conventional log analytics tools provide rudimentary abilities to collect and analyze log records. However, conventional systems cannot efficiently scale when posed with the problem of massive systems involving large numbers of computing systems having large numbers of applications running on those systems. This is because conventional systems often work on a per-host basis, where set-up and configuration activities need to be performed each and every time a new host is added or newly configured in the system, or even where new log collection/configuration activities need to be performed for existing hosts. This approach is highly inefficient given the extensive number of hosts that exist in modern systems. Furthermore, the conventional approaches, particularly on-premise solutions, also fail to adequately permit sharing of resources and analysis components. This causes significant and excessive amounts of redundant processing and resource usage.

Furthermore, conventional log analytics tools also do not provide efficient approaches to classify unknown log types. In many cases, a highly manual process is needed to properly classify the specific type of a given log file. In other cases, automated tools that purport to automatically classify log file types are not accurate enough for modern business environments.

Some embodiments of the invention provide a method and system to implement machine learning-based classification of logs. This approach can be used to group logs automatically using a machine learning infrastructure. Other additional objects, features, and advantages of the invention are described in the detailed description, figures, and claims.

BRIEF DESCRIPTION OF FIGURES

Various embodiments are described hereinafter with reference to the figures. It should be noted that the figures are not drawn to scale and that the elements of similar structures or functions are represented by like reference numerals throughout the figures. It should also be noted that the figures are only intended to facilitate the description of the embodiments. They are not intended as an exhaustive description of the invention or as a limitation on the scope of the invention.

FIG. **1**A illustrates an example system which may be employed in some embodiments of the invention.

FIG. **1**B illustrates a flowchart of a method which may be employed in some embodiments of the invention.

FIG. **2** illustrates a reporting UI.

FIGS. **3**A-C provide more detailed illustrations of the internal structure of the log analytics system and the components within the customer environment that interact with the log analytics system.

FIGS. **4**A-C illustrate approaches to implement the log collection configuration.

FIG. **5** shows a flowchart of an approach to implement a log collection configuration by associating a log rule with a target.

FIG. **6** shows a flowchart of an approach to implement a log collection configuration by associating a log source with a target.

FIG. **7** shows a flowchart of an approach to implement target-based configuration for log monitoring.

FIG. **8** shows a more detailed flowchart of an approach to implement target-based configuration for log monitoring according to some embodiments of the invention.

FIG. **9** illustrates example XML configuration content according to some embodiments of the invention.

FIG. **10** illustrates server-side information to be included in the configuration file to facilitate the log parsing.

FIG. **11** shows a flowchart of one possible approach to implement this aspect of some embodiments of the invention.

FIG. **12** illustrates an architecture for implementing some embodiments of the inventive approach to associate log analysis rules to variable locations.

FIG. **13** illustrates extraction of additional data that is not consistent across all log entries.

FIG. **14** shows some example field definitions.

FIG. **15** shows an architecture for performing machine learning-based classification of logs according to some embodiments of the invention.

FIG. **16** shows a high level flowchart of an approach to implement machine learning-based classification of logs according to some embodiments of the invention.

FIG. **17** illustrates a flowchart of an approach to implement the learning phase according to some embodiments of the invention.

FIG. **18** illustrates an approach for organizing the known set of log data within the log analytics system.

FIGS. **19-1** through **19-5** illustrate the process for generating the learning model for the different log types.

FIG. **20** shows a flowchart of an approach to implement classification according to some embodiments of the invention.

FIGS. **21-1** through **21-11** illustrate a classification process.

FIG. **22** shows a flowchart of an approach to perform processing to adjust centroids.

FIG. **23** illustrates adjustment of a similarity radius.

FIG. **24** illustrates adjustment to a number of centroids.

FIG. **25** shows a flowchart of an approach that can be taken to identify common and variable parts for classification.

FIGS. **26-1** through **26-3** illustrate identification of common and variable parts for classification.

FIG. **27** shows a flowchart of an approach that can be taken to identify field rule types for classification.

FIGS. **28-1** through **28-4** illustrate using field rule types for classification.

FIG. **29** shows an architecture of an example computing system with which the invention may be implemented.

DETAILED DESCRIPTION

As noted above, many types of computing systems and applications generate vast amounts of data pertaining or resulting from operation of that computing system or application. These vast amounts of data are then stored into collected locations, such as log files/records, which can be reviewed at a later time period if there is a need to analyze the behavior or operation of the system or application.

Some embodiments of the invention provide a method and system to implement machine learning-based classification of logs. This approach can be used to group logs automatically using a machine learning infrastructure.

While the below description may describe the invention by way of illustration with respect to "log" data, the invention is not limited in its scope only to the analysis of log data, and indeed is applicable to wide range of data types. Therefore, the invention is not to be limited in its application only to log data unless specifically claimed as such. In addition, the following description may also interchangeably refer to the data being processed as "records" or "messages", without intent to limit the scope of the invention to any particular format for the data.

Log Analytics System

This portion of the disclosure provides a description of a method and system for implementing high volume log collection and analytics, which is usable in conjunction with machine learning classification of log files.

FIG. 1A illustrates an example system **100** for configuring, collecting, and analyzing log data according to some embodiments of the invention. System **100** includes a log analytics system **101** that in some embodiments is embodied as a cloud-based and/or SaaS-based (software as a service) architecture. This means that log analytics system **101** is capable of servicing log analytics functionality as a service on a hosted platform, such that each customer that needs the service does not need to individually install and configure the service components on the customer's own network. The log analytics system **101** is capable of providing the log analytics service to multiple separate customers, and can be scaled to service any number of customers.

Each customer network **104** may include any number of hosts **109**. The hosts **109** are the computing platforms within the customer network **104** that generate log data as one or more log files. The raw log data produced within hosts **109** may originate from any log-producing source. For example, the raw log data may originate from a database management system (DBMS), database application (DB App), middleware, operating system, hardware components, or any other log-producing application, component, or system. One or more gateways **108** are provided in each customer network to communicate with the log analytics system **101**.

The system **100** may include one or more users at one or more user stations **103** that use the system **100** to operate and interact with the log analytics system **101**. The user station **103** comprises any type of computing station that may be used to operate or interface with the log analytics system **101** in the system **100**. Examples of such user stations include, for example, workstations, personal computers, mobile devices, or remote computing terminals. The user station comprises a display device, such as a display monitor, for displaying a user interface to users at the user station. The user station also comprises one or more input devices for the user to provide operational control over the activities of the system **100**, such as a mouse or keyboard to manipulate a pointing object in a graphical user interface to generate user inputs. In some embodiments, the user stations **103** may be (although not required to be) located within the customer network **104**.

The log analytics system **101** comprises functionality that is accessible to users at the user stations **101**, e.g., where log analytics system **101** is implemented as a set of engines, mechanisms, and/or modules (whether hardware, software, or a mixture of hardware and software) to perform configuration, collection, and analysis of log data. A user interface (UI) mechanism generates the UI to display the classification and analysis results, and to allow the user to interact with the log analytics system.

FIG. **1**B shows a flowchart of an approach to use system **100** to configure, collect, and analyze log data. This discussion of FIG. **1**B will refer to components illustrated for the system **100** in FIG. **1**A.

At **120**, log monitoring is configured within the system. This may occur, for example, by a user/customer to configure the type of log monitoring/data gathering desired by the user/customer. Within system **101**, a configuration mechanism **129** comprising UI controls is operable by the user to select and configure log collection configuration **111** and target representations **113** for the log collection configuration.

As discussed in more detail below, the log collection configuration **111** comprise the set of information (e.g., log rules, log source information, and log type information) that identify what data to collect (e.g., which log files), the location of the data to collect (e.g., directory locations), how to access the data (e.g., the format of the log and/or specific fields within the log to acquire), and/or when to collect the data (e.g., on a periodic basis). The log collection configuration **111** may include out-of-the-box rules that are included by a service provider. The log collection configuration **111** may also include customer-defined/customer-customized rules.

The target representations **113** identify "targets", which are individual components within the customer environment that that contain and/or produce logs. These targets are associated with specific components/hosts in the customer environment. An example target may be a specific database application, which are associated with one or more logs one or more hosts.

The ability of the current embodiment to configure log collection/monitoring by associating targets with log rules and/or log sources provides unique advantages for the invention. This is because the user that configures log monitoring does not need to specifically understand exactly how the logs for a given application are located or distributed across the different hosts and components within the environment. Instead, the user only needs to select the specific target (e.g., application) for which monitoring is to be performed, and to then configure the specific parameters under which the log collection process is to be performed.

This solves the significant issue with conventional systems that require configuration of log monitoring on a per-host basis, where set-up and configuration activities need to be performed each and every time a new host is added or newly configured in the system, or even where new log collection/configuration activities need to be performed for existing hosts. Unlike conventional approaches, the log analytics user can be insulated from the specifics of the exact hosts/components that pertain to the logs for a given target. This information can be encapsulated in underlying metadata that is maintained by administrators of the system that understand the correspondence between the applications, hosts, and components in the system.

The next action at **122** is to capture the log data according to the user configurations. The association between the log rules **111** and the target representations is sent to the customer network **104** for processing. An agent of the log analytics system is present on each of the hosts **109** to collect data from the appropriate logs on the hosts **109**.

In some embodiments, data masking may be performed upon the captured data. The masking is performed at collection time, which protects the customer data before it leaves the customer network. For example, various types of information in the collected log data (such as user names and other personal information) may be sensitive enough to be masked before it is sent to the server. Patterns are identified for such data, which can be removed and/or changed to proxy data before it is collected for the server. This allows the data to still be used for analysis purposes, while hiding the sensitive data. Some embodiments permanently remove the sensitive data (e.g., change all such data to "***" symbols), or changed to data that is mapped so that the original data can be recovered.

At **124**, the collected log data is delivered from the customer network **104** to the log analytics system **101**. The multiple hosts **109** in the customer network **104** provide the collected data to a smaller number of one or more gateways **108**, which then sends the log data to edge services **106** at the log analytics system **101**. The edge services **106** receives the collected data one or more customer networks and places the data into an inbound data store for further processing by a log processing pipeline **107**.

At **126**, the log processing pipeline **107** performs a series of data processing and analytical operations upon the collected log data, which is described in more detail below. At **128**, the processed data is then stored into a data storage device **110**. The computer readable storage device **110** comprises any combination of hardware and software that allows for ready access to the data that is located at the computer readable storage device **110**. For example, the computer readable storage device **110** could be implemented as computer memory operatively managed by an operating system. The data in the computer readable storage device **110** could also be implemented as database objects, cloud objects, and/or files in a file system. In some embodiments, the processed data is stored within both a text/indexed data store **110***a* (e.g., as a SOLR cluster) and a raw/historical data store **110***b* (e.g., as a HDFS cluster).

At **130**, reporting may be performed on the processed data using a reporting mechanism/UI **115**. As illustrated in FIG. **2**, the reporting UI **200** may include a log search facility **202**, one or more dashboards **204**, and/or any suitable applications **206** for analyzing/viewing the processed log data. Examples of such reporting components are described in more detail below.

At **132**, incident management may be performed upon the processed data. One or more alert conditions can be configured within log analytics system such that upon the detection of the alert condition, an incident management mechanism **117** provides a notification to a designated set of users of the incident/alert.

At **134**, a Corrective Action Engine **119** may perform any necessary actions to be taken within the customer network **104**. For example, a log entry may be received that a database system is down. When such a log entry is identified, a possible automated corrective action is to attempt to bring the database system back up. The customer may create a corrective action script to address this situation. A trigger may be performed to run the script to perform the corrective action (e.g., the trigger causes an instruction to be sent to the agent on the customer network to run the script). In an alternative embodiment, the appropriate script for the situation is pushed down from the server to the customer network to be executed. In addition, at **136**, any other additional functions and/or actions may be taken as appropriate based at last upon the processed data.

FIG. **3**A provides a more detailed illustration of the internal structure of the log analytics system at a host environment **340** and the components within the customer environment **342** that interact with the log analytics system. This architecture **300** is configured to provide a flow for log monitoring that is able to handle large amounts of log data ingest.

In the customer environment **342** within a single customer host/server **344**, the LA (log analytics) agent **333** takes the log monitoring configuration data **332** (e.g., sniffer configuration or target-side configuration materials), and calls a log file **336** sniffer (also referred to herein as the "log collector") to gather log data from one or more log files **338**. A daemon manager **334** can be employed to interface with the log file sniffer **336**. The log file sniffer **336** reads from one or more log files **338** on the host machine **344**. The daemon manager **334** takes the log content and packages it up so that it can be handed back to the LA agent **333**. It is noted that the system may include any number of different kinds of sniffers, and a log sniffer **336** is merely an example of a single type of sniffer that can be used in the system. Other types of sniffers may therefore be employed within various embodiments of the invention, e.g., sniffers to monitor registries, databases, windows event logs, etc. In addition, the log sniffer in some embodiments is configured to handle collective/compressed files, e.g., a Zip file.

The LA agent **333** sends the gathered log data to the gateway agent **330**. The gateway agent **330** packages up the log data that is collected from multiple customer hosts/servers, essentially acting as an aggregator to aggregate the log content from multiple hosts. The packaged content is then sent from the gateway agent **330** to the edge services **306**. The edge services **306** receive a large amount of data from multiple gateway agents **330** from any number of different customer environments **342**.

Given the potentially large volume of data that may be received at the edge services **306**, the data is immediately stored into an inbound data storage device **304** (the "platform inbound store"). This acts as a queue for the log processing pipeline **308**. A data structure is provided to manage the items to be processed within the inbound data store. In some embodiments, a messaging platform **302** (e.g., implemented using the Kafka product) can be used to track the to-be-processed items within the queue. Within the log processing pipeline **308**, a queue consumer **310** identifies the next item within the queue to be processed, which is then retrieved from the platform inbound store. The queue consumer **310** comprises any entity that is capable of processing work within the system off the queue, such as a process, thread, node, or task.

The retrieved log data undergoes a "parse" stage **312**, where the log entries are parsed and broken up into specific fields. As discussed in more detail below, the "log type" configured for the log specifies how to break up the log entry into the desired fields.

In the "normalize" stage **314**, the identified fields are normalized. For example, a "time" field may be represented in any number of different ways in different logs. This time field can be normalized into a single recognizable format (e.g., UTC format). As another example, the word "error" may be represented in different ways on different systems (e.g., all upper case "ERROR", all lower case "error", first letter capitalized "Error", or abbreviation "err"). This situation may require the different word forms/types to be normalized into a single format (e.g., all lower case unabbreviated term "error").

The "transform" stage **316** can be used to synthesize new content from the log data. As an example and which will be discussed in more detail below, "tags" can be added to the log data to provide additional information about the log entries. As another example, field extraction can be performed to extract additional fields from the existing log entry fields.

A "condition evaluation" stage **318** is used to evaluate for specified conditions upon the log data. This stage can be performed to identify patterns within the log data, and to create/identify alerts conditions within the logs. Any type of notifications may be performed at this stage, including for example, emails/text messages/call sent to administrators/customers or alert to another system or mechanism.

A log writer **320** then writes the processed log data to one or more data stores **324**. In some embodiments, the processed data is stored within both a text/indexed data store (e.g., as a SOLR cluster) and a raw and/or historical data store (e.g., as a HDFS cluster). The log writer can also send the log data to another processing stage **322** and/or downstream processing engine.

As shown in FIG. **3**B, some embodiments provide a side loading mechanism **350** to collect log data without to proceed through an agent **333** on the client side. In this approach, the user logs into the server to select one or more files on a local system. The system will load that file at the server, and will sniff through that file (e.g., by having the user provide the log type, attempting likely log types, rolling through different log types, or by making an educated "guess" of the log type). The sniffing results are then passed to the Edge Services and process as previously described. In the embodiment, of FIG. **3**C, only the side loading mechanism **350** exists to gather the log files—where the agent/sniffer entities are either not installed and/or not needed on the client server **344**.

FIGS. **4**A-B illustrate approaches to implement the log collection configuration. This approach allow for very large scale configuration of how to monitor log files having one or more log entries. In some embodiments, a log entry corresponds to a single logical row from a log file. In the actual log file, a single entry could take multiple lines due to carriage returns being part of the log entry content. This entire content is considered a single "entry". Each entry starts with "####<date" and could occupy a single physical line in the file or multiple lines separate by carriage returns.

In this model the "Log Type" **406** defines how the system reads the log file, as well as how to decompose the log file into its parts. In some embodiments, a log file contains several base fields. The base fields that exist may vary for different types of logs. A "base parser" can be used to breaks a log entry into the specified fields. The base parser may also perform transformations. For instance, a Date field can be converted to a normalized format and time adjusted to be in UTC so data from many locations can be mixed together.

The "Log Source" **404** defines where log files are located and how to read them. In some embodiments, the log source is a named definition that contains a list of log files described using patterns, along with the parser that is needed to parse that file. For instance, one source could be "SSH Log files". This source may list each log file related to SSH separately, or could describe the log files using a wildcard (e.g., "/var/log/ssh*"). For each pattern, a base parser can be chosen (e.g., by a user) to parse the base fields from the file. This approach can be used to ensure that for a single pattern that all files conform to the same base parse structure. For one source, one can choose from among multiple log types, and give a priority to those possible types. For example, types A, B, and C can be identified, where the analysis works through each of these in order to determine whether the source matches one of these identified types. Therefore, for each pattern, the user can choose multiple base parsers. In some embodiments, the same source may match against and be analyzed using multiple types.

The "Log Rule" **402** defines a set of sources along with conditions and actions to be triggered during continuous monitoring. The "Targets" **408** identify individual components in an IT environment that contain logs. Associating a rule to a target starts the monitoring process in some embodiments.

In the embodiment of FIG. **4**A, one or more log rules are associated with one or more targets. In the alternative embodiment of FIG. **4**B, one or more log sources can be associated with one or more targets to create an instance of a target. In the embodiment of FIG. **4**C, log rules are not even provided as an approach to create the associations—where only log source to target associations are provided to create target instances. Each of these approaches are described in more detail below.

FIG. **5** shows a flowchart of an approach to implement a log collection configuration by associating a log rule with a target. At **502**, one or more log rules are created. The rules are processed by a rules engine within the log processing system to implement rule-based handling of a given target. Therefore, the rule will include specific logic for handling a given target that it is associated with.

In some embodiments, the rule can be used to specific a target type, which identifies the type of the target that the rule is intended to address. A rule can be specified for a single target type or multiple target types. For example, when monitoring a log file for a database instance, the target type can be set to Database Instance so that reporting of activities in the log goes against the proper target type; In some embodiments, even though the rule may be configured for a "File" as a log type, the target type can still be any managed target type, such as a database.

The rule may specify a source type, which identifies the type of log file that the rule is intended to address. For example the rule may specify that the log file types will be: (i) File: OS level log file; (ii) Database Table: a table that stores log content in a database; (iii) Windows Event Log: read events from windows event as log content.

A target property filter may be specified in the rule to filter for targets to specify conditions under which the rule is applicable, such as for example, a particular operating system (OS), target version, and/or target platform. For instance, the user could create a rule that is only for a given OS on a given platform (e.g., only for Linux OEL5 on X86_64 hardware).

When creating rules in some embodiments, the rule the may also include: (a) the name of the rule; (b) a severity level indicating how important the outcome of this rule is if this rule leads to an event being generated; (c) a description of the rule; and/or (d) a textual rationale of why this monitoring is occurring.

In some embodiments, one or more conditions can be established for which the rule will "trigger". Multiple conditions may be specified, where each condition can be combined with others using a Boolean operator. For example, a set of conditions that is ORed with others means that if any of these conditions match an entry in a log file under evaluation, then that entry triggers this rule. When the conditions are ANDed together, all clauses of the condition must be met for the condition to trigger an entry in a log file. The specified actions will then be taken as a response to this entry that is matched. The following is an example condition clause that includes a regular expression: "MESSAGE contains "START: telnet pid=[0-9]* from=[.]*"", where this condition triggers the rule if the message matches the regular expression.

The "operator" in the condition is how the comparison is to be performed. The following are some example operators that may be employed in some embodiments of the invention: (a)<, >, >=, <=: compare a value to be larger or smaller (or equal) than some set value; (b) Contains: pattern match with ability to include regular expression clauses, where an implicit wildcard may be placed at the beginning and end unless the user uses the ^ and $ regular expression symbols to specify the beginning of a string or end of the string; (c) In: list of possible values; (d) Is: exact string match (no regular expression capability); (e) Is Not; (f) Does Not Contain; (g) Not In: List of values to not match.

Actions may be specified to identify what to do when a match is found on the selected sources for a given condition. For example, one possible action is to capture a complete log entry as an observation when matching conditions of the rule. This approach lets the system/user, when monitoring a log from any source and when a single entry is seen that matches the conditions of this rule, to save that complete entry and store it in the repository as an observation. Observations are stored for later viewing through the log observations UI or other reporting features. Another possible action is to create an event entry for each matching condition. When a log entry is seen as matching the specified conditions, this approaches raise an event. In some embodiments, the event will be created directly at the agent. The source definition will define any special fields that may be needed for capturing events if there are any. An additional option for this action is to have repeat log entries bundled at the agent and only report the event at most only once for the time range the user specified. The matching conditions can be used to help identify the existence of a repeat entry. Another example action is to create a metric for the rule to capture each occurrence of a matching condition. In this approach, a new metric is created for this rule using a metric subsystem. Thereafter, when there is a log entry that matches the rule's conditions, some number of the fields are captured as metric data and uploaded as part of this metric. The fields can be selected to include, for example, information such as "key" fields like target, time, source, etc.

At **504**, one or more targets are identified in the system. The targets are individual components within the customer environment that that contain logs. These targets are associated with specific components/hosts in the customer environment. Example targets include hosts, database application, middleware applications, and/or other software applications, which are associated with one or more logs one or more hosts. More details regarding an approach to specify targets are described below.

At **506**, an association is made between a target and a rule. Metadata may be maintained in the system to track the associations between a given target and a given rule. A user interface may be provided that allows a user to see what targets a selected rule is associated with and/or to add more associations, where the associations are the way the rule becomes active by associating the rule against a real target.

Thereafter, at **508**, log collection and processing are performed based at least in part upon the association between the rule and the target. As discussed in more detail below, target-based configuration may involve various types of configuration data that is created at both the server-side and the target-side to implement the log collection as well as log processing.

The ability of the current embodiment to configure log collection/monitoring by associating targets with log rules provides unique advantages. This is because the user that configures log monitoring does not need to specifically understand exactly how the logs for a given application are located or distributed across the different hosts and components within the environment. Instead, the user only needs to select the specific target (e.g., application) for which monitoring is to be performed and to then configure the rules under which the log collection process is to be performed.

This solves the significant issue with conventional systems that require configuration of log monitoring on a per-host basis, where set-up and configuration activities need to be performed each and every time a new host is added or newly configured in the system, or even where new log collection/configuration activities need to be performed for existing hosts. Unlike conventional approaches, the log analytics user can be insulated from the specifics of the exact hosts/components that pertain to the logs for a given target. This information can be encapsulated in underlying metadata that is maintained by administrators of the system that understand the correspondence between the applications, hosts, and components in the system.

Instead of, or in addition to the rules, log processing can also be configured by associating a log source to a target. FIG. **6** shows a flowchart of an approach to implement a log collection configuration by associating a log source with a target. At **602**, one or more log sources are created. The log source defines where log files are located and how to read them. The log source may define a source type that indicates how the source content is gathered. The following are example source types: (a) File—identifies a readable file from the OS level that can be accessed using regular OS-level file operations; (b) Database Table—a table that stores log entries (e.g.: database audit table); (c) Windows Event System—an API that provides access to event records. One or more source names may be defined for the log source. In addition, the log source may be associated with a description of the source. It is noted that log sources can also be used when creating log monitoring rules (as described above).

The log source may also be associated with a file pattern and/or pathname expression. For instance, "/var/log/messages*" is an example of a file pattern (that may actually pertain to a number of multiple files). Regarding file patterns, one reason for their use in the present log analytics system is because it is possible that the exact location of the logs to monitor varies. Some of the time, a system will expect logs to be in a particular place, e.g., in a specific directory. When the system is dealing with a large number of streaming logs, it may not be clear which directory the logs are expected to be in. This prevents a system that relies upon static log file locations to operate correctly. Therefore, the file pattern is useful to address these possibly varying log locations.

In some embodiments, a log source is created by specifying a source name and description for the log source. The definition of the log source may comprise included file name patterns and excluded file name patterns. The file name patterns are patterns that correspond to files (or directories) to include for the log source. The excluded file name patterns correspond to patterns for files (or directories) to explicitly exclude from the log source, e.g., which is useful in the situation where the included file name pattern identifies a directory having numerous files, and some of those files (such as dummy files or non-log files) are excluded using the excluded file name pattern. For each pattern, the system captures the pattern string, the description, and the base parser (log type) that will be used to parse the file. The base parser may define the basic structure of the file, e.g., how to parse the data, hostname, and message from the file.

The definition of the log source may also specify whether the source contains secure log content. This is available so that a source creator can specify a special role that users must have to view any log data may be captured. This log data may include security-related content that not any target owner can view.

As noted above, the log rules may reference log sources, and vice versa. In some embodiments, the system metadata tracks these associations, so that a count is maintained of rules that are currently using sources. This helps with understanding the impact if a source and/or rule is changed or deleted.

At **604**, one or more targets are identified. As noted above, targets are components within the environment that that contain, correspond, and/or create logs or other data to be processed, where the targets are associated with specific components/hosts in the customer environment. Example targets include hosts, database application, middleware applications, and/or other software applications, which are associated with one or more logs one or more hosts.

At **606**, an association is made between a target and a source. Metadata may be maintained in the system to track the associations between a given target and a given source. A user interface may be provided that allows a user to see what targets a selected source is associated with and/or to add more associations.

The association of the target to the source creates, at **608**, a specific instance of the log source. For example, consider a log source that generically specifies that a given file is located at a given directory location (e.g., c:/log_directory/log_file). It may be the case that any number of servers (Server A, Server B, Server C, Server D) within a customer environment may have a copy of that file (log_file) in that directory (c:/log directory). However, by associating a specific target (e.g., Server A) to the log source, this creates an instance of the log source so that the new instance is specific regarding the log file in the specified directory on a specific target (e.g., to begin monitoring c:/log_directory/log_file specifically on Server A).

Thereafter, at **610**, log collection and processing are performed based at least in part upon the association between the rule and the log source. As discussed in more detail below, target-based configuration may involve various types of configuration data that is created at both the server-side and the target-side to implement the log collection and processing activities.

There are numerous benefits when using this type of model for configuring log collection. One benefit is that the Log Types, Sources, Rules can be easily reused as necessary. In addition, this approach avoids having to make numerous duplicate configurations by enabling sharing at multiple levels. Moreover, users can create custom rules that use sources and log types defined by other people or ship with the product. This approach also easily builds on top of shared knowledge.

Associating rules/sources to targets provides knowledge that identifies where to physically enable log collections via the agents. This means that users do not need to know anything about where the targets are located. In addition, bulk association of rules/sources to targets can be facilitated. In some embodiments, rules/sources can be automatically associated to all targets based on the configuration. As noted above, out-of-the-box configurations can be provided by the service provider. In addition, users can create their own configurations, including extending the provided out-of-the-box configurations. This permits the users to customize without building their own content.

FIG. **7** shows a flowchart of an approach to implement target-based configuration for log monitoring. This process generates the creation, deployment, and/or updating of configuration materials for log monitoring. In some embodiments, configuration materials are embodied as configuration files that are used by the log monitoring system to manage and implement the log monitoring process.

At **700**, target-based processing is initiated. Example approaches for initiating target-based processing includes, for example, installation of a log analytics agent onto a specific log collection location. The target-based processing pertains to associations made between one or more targets and one or more log sources and/or rules.

At **702**, configuration materials are generated for the target-based processing. In some embodiment, the target-based configuration file is implemented as configuration XML, files, although other formats may also be used to implement the configuration materials. The target-based configuration file may be created at a master site (e.g., to create a master version **704**), with specific versions then passed to both the server side and the target side.

The target-side materials **708** may comprise those portions of the configuration details that are pertinent for log collection efforts. This includes, for example, information about log source details and target details. The server-side materials **706** may comprise portions of the configuration details that are pertinent to the server-side log processing. This includes, for example, information about parser details.

In some embodiments, a database at the server maintains a master version and a target version of the configuration materials. As noted above, the target version includes configuration details that are pertinent to log collection efforts, and is passed to the customer environment to be used by the agent in the customer environment to collect the appropriate log data from the customer environment. The master version includes the full set of configuration details needed at the server, and becomes the "server side" materials when selected and used for processing at the server. This may occur, for example, when the log data collected at the targets are passed to the server, where the transmission of the log data includes an identifier that uniquely identifies the target-side materials used to collect the log data (e.g., the configuration version or "CV" number **903** shown in the example targets-side materials of FIG. **9**). When this data is received at the server, the identifier is used to determine the corresponding master version of the materials that have the same identifier number (e.g., as shown in field **1003** in the example server-side materials of FIG. **10**). That master version is then used as the server-side materials to process the received log data. Therefore, in this embodiment, the master version **704** and the server-side materials **706** are identical, but having different labels depending upon whether the material is currently in-use to process the log data. In an alternative embodiment, the master version may differ from a server version, e.g., where the materials are used on multiple servers with different configuration details.

At **710**, the configuration materials are then distributed to the appropriate locations within the log processing system. In some embodiments, the target-side materials **708** are distributed to the customer system as the sniffer configuration files **332** shown in FIG. **3**A. With regards to the server-side materials **706**, the materials are "distributed" as the log configuration files **111** shown in FIG. **1**A, where the distribution does not actually require the materials to be distributed across a network, but merely indicates that the materials are obtained from another component within the server (e.g., on an as-needed basis).

Thereafter, at **712**, log collection processing is performed at the target using the target-side configuration materials. In addition, at **714**, server-side log processing is performed using the server-side configuration materials.

FIG. **8** shows a more detailed flowchart of an approach to implement target-based configuration for log monitoring according to some embodiments of the invention. At **802**, one or more work items for processing target associations are created in the system. For example, this type of work may be created upon installation of the log analytics agent onto a target, where recognition of this installation causes a work item to be created for the target-based configuration materials. A list of target types are identified that have at least one auto-association rule (e.g., from a database of the associations). A list of targets is generated for which there is a need to be associated with auto-enabled rules. These steps are equivalent to putting association tasks into a queue (e.g., database table) by a producer entity/process, which are then processed by one or more consumer entities/processes.

One or more consumer/worker entities may wake up periodically to process the work items. For example, a worker entity (e.g., thread or process) wakes up (e.g., every 10 seconds) to check whether there are any pending association tasks. The set of one or more workers will iterate through the tasks to process the work in the queue.

At **804**, one of the workers identifies an association task to process. At **806**, the association request is processed by accessing information collected for the rules, sources, parsers, fields, and/or target. This action identifies what target is being addressed, finds that target, and then looks up details of the log source and/or log rule that has been associated with the target.

At **808**, the worker then generate configuration content for the specific association task that it is handling. In some embodiments, the configuration content is embodied as XML content. This action creates both the target-side details and the server-side details for the configuration materials. For the server-side, this action will create configuration data for the server to process collected log data. For example, parser details in XML, format are created for the server-side materials for the log data expected to be received. For the target-side, this action will create configuration data for log collection from the target. For example, as discussed below, variable pathnames (e.g., having variables instead of absolute pathnames) may be specified for a given log source to identify a directory that contains log files to monitor. These varying pathnames may be replaced with actual pathnames and inserted into the target-side materials at step **808**.

A determination is made at **810** whether there are any additional association tasks to process. If there are additional tasks on the queue, then the process returns back to **804** to select another task to process. If not, then at **812**, the configuration materials are finalized.

It is noted that the same configuration/XML file can be used to address multiple associations. For example, if multiple targets are on the same host, then a single configuration file may be generated for all of the targets on the host. In this case, step **808** described above appends the XML, content to the same XML, file for multiple iterations through the processing loop.

Updates may occur in a similar manner. When a change occurs that requires updating of the materials, then one or more new association tasks may be placed onto a queue and addressed as described above. Furthermore, de-associations may also occur, e.g., where the log analytics agent is de-installed. In this situation, the configuration files may be deleted. When a target is deleted, a message may be broadcast to notify all listeners about this event by a target model service, which may be consumed to delete the corresponding associations and to update the XML content.

FIG. **9** illustrates example XML configuration content **900** according to some embodiments of the invention. This is an example of target-side content that may be placed on the host that holds the target. This XML configuration content **900** defines a rule to collect Linux system message logs with file pattern "/var/log/messages*" on host XYZ.us.oracle.com. Portion **902** identifies a base parser for the association being addressed. Portion **903** provides an identifier for the version number ("configuration version" or "CV") of the content **900**, which is used to match up against the corresponding server-side materials having the same version number. Portion **904** identifies the ID of a log rule. Portion **906** identifies a specific target. Portion **908** identifies a target type. Portion **910** identifies a source type. Portion **912** identifies a parser ID for the source. The logs will be parsed based on some defined parser. Such configuration files reside on sniffers and the log collection processes collect logs based on the defined log sources.

In the log processor at the server side, additional information can be included in the configuration file to facilitate the log parsing, e.g., as shown in the server-side content portion **1000** of FIG. **10**. The FieldDef portion **1001** indicates the data type for the service. The Log Source portion **1002** indicates the logs are of "os_file" type. The BaseParse portion **1004** defines the way to parse the log entries based on defined regular expressions in portion **1006**. Portion **1003** provides an identifier for the version number of the content **1000**, which is used to match up against the corresponding target-side materials having the same version number.

In addition to the above-described auto-associations, target-source manual associations may also be performed. For example, a user interface may be provided to perform the manual associations. This also causes the above-described actions to be performed, but is triggered by the manual actions.

Re-synchronization may be performed of target-source associations. To explain, consider that when a log analytics agent is installed, monitored targets connected through the agent can be associated with certain pre-defined log sources Similarly, when the agent is de-installed, such associations can be deleted from the appropriate database tables. In addition, when a target is added to be monitored by an agent, the target can be associated with certain pre-defined log sources for that target type, and when the target is deleted from an agent, such association can be deleted from database tables.

Over time, these associations could become out-of-sync due to various reasons. For example, when a log analytics agent is being installed, the auto-association may occur due to some network issue that causes the loss of the configuration materials during its transfer. In addition, when a target is added or deleted, an event may not processed properly so the configuration XML, file when updating does not occur as appropriate.

To handle these cases and maintain the association consistency between targets and their corresponding log sources, a web service is provided in some embodiments to synchronize the associations periodically. In at least one embodiment, only the auto-associations are synched, and not the manual associations customized by users manually.

Associations may be performed for a specific log analytics agent. A delta analysis can be performed between targets in a data model data store and targets in a log analytics data store to implement this action. Processing may occur where: (a) For targets in data model data store but not in log analytics data store, add associations for these targets; (b) For targets not in data model data store but in log analytics data store, delete associations for these targets; (c) For targets in data model data store and log analytics data store, keep the same associations for these targets in case of user customization. One potential issue for adding associations pertains to the situation where a user may have deleted all associations for a particular target so there is no entry in the log analytics data store, but there is an entry in the data model data store. The issue is that when applying the above approach, the auto-associations not wanted could be brought in again after the synchronization operation. To avoid this, the system can record the user action to identify the potential issue.

In addition, associations may be synchronized for a specified tenant. When this action is performed, delta analysis can be performed between the agent for the data model data store and agent for the log analytics data store. Processing may occur by: (a) For an agent in the data model data store but not in the log analytics data store, add associations for these agents; (b) For agents not in the data model data store but in the log analytics data store, delete associations for these agents; (c) For agents in the data model data store and the log analytics data store, perform the same delta analysis and synchronization as described above.

Synchronization may be performed for associations for all tenants. When this action is performed, it should perform agent-level synchronization as described for each tenant.

Turning the attention of this document to file patterns, one reason for their use in log analytics systems is because it is possible that the exact location of the logs to monitor varies. Most of the time, a system will expect logs to be in a particular place, in a specific directory. When the system dealing with a large number of streaming logs, it may not be clear which directory the logs are expected to be in. This prevents a system that relies upon static log file locations from operating correctly.

The inventive approach in some embodiments can associate log analysis rules to variable locations. One approach is to use metadata that replaces variable parts that correspond to locations for the log files. A path expression is used to represent the pathname for the log files, where the path expression includes a fixed portion and a varying portion, and different values are implemented for the variable part. The placeholder for location is eventually replaced with the actual location in the directory path.

Some embodiments provide for "parameters", which are flexible fields (e.g., text fields) that users can use in either the include file name patterns or exclude file name patterns. The parameters may be implemented by enclosing a parameter name in curly brackets {and}. A user-defined default value is provided in this source. A user can then provide a parameter override on a per target basis when associating a log monitoring rule using this source to a target. The overrides are particularly applicable, for example, with regards to changes from out-of-the-box content (e.g., to override rules, definitions, etc. without actually changing the OOTB content). This is implemented, for example, by implementing a mapping/annotation table that includes the user overrides and indicate of an override for the OOTB content.

The reason this is very helpful is because in the log sources, paths may be defined for log files to monitor. In some cases, the paths are fixed, such as in the Linux syslog file, the path is "/var/log/messages*". However, in other cases, one may want to monitor a database alert log, where each database target will be installed in a completely different path, and the path to find the alert log may be different. For example, the alert log for one database is located at this location: "/xxx/db/yyyy/oracle/diag/rdbms/set2/set2/alert/log*.xml". The underlined portions may vary for every database target. However, each target has the notion of target properties. Included in these properties are metadata that can be used to fill in the variable parts in the path. In the current embodiment, one can express this path instead as: "{DIAGNOSTIC_DEST}/diag/rdbms/{SID}/{SID}/alert/log*.xml"

When this source is used in a rule and this rule is associated to the target, the system replaces the parameters "DIAGNOSTIC_DEST" and "SID" with those that are known for that target. This allows the system to associate a single rule and source to thousands of targets at once.

As another example, the user may want to monitor the pattern: "/xxx/oracle/log/*". In this case, "/xxx/oracle" is a variable path depending on the host. One could instead write the pattern as: "{INSTALL_DIR}/log/*". For this source, the user can provide a default value (/xxx/oracle) to the INSTALL_DIR parameter. Later, when rule is associated to a target, the user can provide a target override value of "/xxx/oracle" for this parameter on this target without having to create a new source or rule.

With regards to system-defined fixed parameters, there may be a case where the user wishes to reference a built-in parameter (e.g., ORACLE_HOME). Here, the system will replace that variable with the ORACLE_HOME that is known for the selected target. The pattern could be written as: "{ORACLE_HOME}/log/*". This path will automatically be understood by the agent, where ORACLE_HOME is a special built-in parameter that does not need a default to be set by the user. The system could be provided with a list of fixed parameters that integrators/users can choose to use.

FIG. **11** shows a flowchart of one possible approach to implement this aspect of some embodiments of the invention. At **1102**, identification is made of location content for which it is desirable to implement variable location processing. This situation may exist, for example, when the system is handling a large number of streaming logs from possibly a large number and/or uncertain of directory locations. The log data may be located at target locations that are addressed using a pathname that varies for different database targets.

At **1104**, a path is specified for the target locations having a fixed portion and a varying portion. The varying portion may be represented with one or more parameters. During log processing, at **1106**, the one or more parameters are replaced with values corresponding to one or more target log files, wherein a single rule for implementing log monitoring is associated with multiple different targets to be monitored.

This approach is quite advantageous over approaches where every log is in a different directory that one cannot know about ahead of time, and where a separate forwarder mechanism would have to be set up for each path. Instead, the present approach can be used to set up one rule for a very large number of paths.

In some embodiments, configuration information from the log analytics system can be coupled to this approach to configure and setup the rules for identifying log file assignments. Some examples of configuration information that can be used include, for example, how a database is connected, how the components are connected, which datacenter is being used, etc.

Some embodiments specify how to map sources to targets based on their relationships. For instance, a defined source Source1 can be assigned to all related targets belonging to a certain system. Any association type and/or rule can be used in this embodiment, e.g., where a common set of association types is used to provide configuration information useful for determining rules for log locations. Such association types may include, for example, "contains", "application_contains", "app_composite_contains", "authenticated_by", "composite_contains (abstract)", "cluster_contains", "connects_through", "contains (abstract)", "depends_on(abstract)", "deployed_on", "exposes", "hosted_by", "installed a "managed_by", "monitored_by", "provided_by", "runs_on (abstract)", "stores_on", stores_on_db" and "uses (abstract)".

It is noted that the target relationship information/model can be used in other ways as well. For example, the target model can also be used to help correlate log entry findings to aid in root cause analysis. As another example, the host model can be used for comparing all hosts in one system. For instance, if there are a number of databases in a first system, this feature can be used to see logs across these systems together, and in isolation from databases used for a second system.

FIG. **12** illustrates an architecture for implementing some embodiments of the inventive approach to associate log analysis rules to variable locations. Here, the log analytics engine **1202** operates by accessing log collection configuration files **1211**. Log collection configuration files **1211** is implemented to represent a path where the target location may have both a fixed portion and a varying portion. The varying portion may be represented with one or more location parameters. In this example, different locations may exist for logs **1201***a*, **1201***b*, and **1201***c*. By replacing the variable portion, the specific location for the log of interest may be selected by the log analytics engine **1202**, and processed to generate analysis results **1213**.

Here, the reference material **1210** may be accessed to identify the correct replacement of the variable portions of the paths for the target locations. Any suitable type of reference materials may be implemented. As noted above, a defined source Source1 can be assigned to all related targets belonging to a certain system, and/or an association type and/or rule can be used as well. In addition, target relationship information/models can be employed as well as the reference material.

Embodiments of the invention therefore provides improved functionality to perform target-based log monitoring. Two possible use cases this functionality includes log monitoring and ad hoc log browsing. Log monitoring pertains, for example, to the situation where there is continuous monitoring and capture of logs. Some embodiments of log monitoring pertains to the some or all of the following: (a) monitor any log for any target and capture significant entries from the logs; (b) create events based on some log entries; (c) identify existence of log entries that can affect a compliance score; (d) perform user as well as integrator defined monitoring; (e) capture log entries that are not events to enable analytics on a subset of all logs; (f) use cases such as intrusion detection, potential security risk detection, problem detection; (g) implement long term persistent storage of log contents; (h) search for log content; (i) customizable search-based views; (j) log anomaly detection and scoring

Ad hoc log browsing pertains, for example, to the situation where there is not continuous monitoring of logs. In this approach, the user can browse live logs on a host without having to collect the logs and send them up to the SaaS server. The model for configuring what to monitor is similar to what was described earlier. The difference pertains to the fact that the user can select a rule, source, and some filters from the UI and the search is sent down to agent to obtain log files that match and bring them back, storing them in a temporary storage in the server. The user can continue to narrow their search down on that result set. If the user adds another target, rule, or extends the time range, the system goes back to the agent to obtain only the delta content, and not the entire content again. The user can therefore get the same benefits of log analytics without configuring continuous log monitoring. The feature can be very low-latency since the system only needs to go back to get more data from agent when the search is expanded. All searches that are narrowing down current result set goes against the data that have been cached from a previous get from the agent.

The embodiments of the invention can be used to store log data into a long-term centralized location in a raw/historical datastore. For example, target owners in the company IT department can monitor incoming issues for all responsible targets. This may include thousands of targets (hosts, databases, middle wares, and applications) that are managed by the SaaS log analytics system for the company. Many log entries (e.g., hundreds of GB of entries) may be generated each day. For compliance reasons, these logs may be required to be stored permanently, and based on these logs, the data center manager may wish to obtain some big pictures of them in long run and IT administrators may wish to search through them to figure out some possible causes of a particular issue. In this scenario, a very large amount of logs could be stored in a centralized storage, on top of which users can search logs and view log trends with acceptable performance. In some embodiments, the log data can be stored in an off-line repository. This can be used, for example, when data kept online for a certain period of time, and then transferred offline. This is particularly applicable when there are different pricing tiers for the different types of storage (e.g., lower price for offline storage), and the user is given the choice of where to store the data. In this approach, the data may held in offline storage may be brought back online at a later point in time.

The logs can be searched to analyze for possible causes of issues. For example, when a particular issue occurs to a target, the target owner can analyze logs from various sources to pinpoint the causes of the issue. Particularly, time-related logs from different components of the same application or from different but related applications could be reported in a time-interleaved format in a consolidated view to help target owner to figure out possible causes of the issue. The target owner could perform some ad-hoc searches to find same or similar log entries over the time, and jump to the interested log entry, and then drill down to the detailed message and browse other logs generated before/after the interested point.

In some embodiments, restrictions can be applied such that users have access only to logs for which access permissions are provided to those users. Different classes of users may be associated with access to different sets of logs. Various roles can be associated with permissions to access certain logs.

Some embodiments can be employed to view long-term log distribution, trends, and correlations. With many logs generated by many different targets and log sources over long time, data center managers may wish to view the long-term log distributions and patterns.

Some embodiments can be employed to search logs to identify causes of an application outage. Consider the situation where an IT administrator or target owner of a web application receives some notification that some customers who used the application reported that they could not complete their online transactions and the confirmation page could not be shown after the submit button was clicked. With embodiments of the invention, the IT administrator can search the logs generated by the application with the user name as key and within the issue reporting time range. Some application exception may be found in the log indicating that some database error occurred when the application tried to commit the transaction. By adding the database and its corresponding hosting server via target association relationship and their availability related log sources for the search, the IT administrator could browse the logs around the application exception time to find some database errors, which was related for example to some hosting server partial disk failure and high volume of committing transactions.

Some embodiments can be employed to view long-term log distributions, trends, and correlations by tags. A data center manager may define some tags for logs collected in the data center, such as security logs for production databases, security logs for development servers, logs for testing servers, noise logs, etc. The data manager may be interested, for example, in knowing the followings: log distributions by these tags over the past half year, their daily incoming rates during last month, and whether there are any correlations between the security log entries for production databases and the changes of their compliance scores during a given time period.

Some embodiments permit log data to be stored as metrics. In certain embodiments, the system will store several log fields as key fields. The key fields will include (but may not be limited to): Time, Target, Rule, Source, and Log File. The system may also create a hash or GUID to distinguish possible log entries that have the same time and all other key fields. When a rule that is using this metric action for log entries is associated with the first target, a metric extension is created and deployed. This metric extension will be named similar to the rule to make it easy for the user to reference it.

In some embodiments, the log monitoring rule has a possible action to create an event when a log entry matches the condition of the rule. Additionally, users will be able to indicate that this event should also trigger a compliance violation which will cause an impact on the compliance score for a compliance standard and framework.

As noted above, one possible use case is to provide a log browser, e.g., where browsing is employed to browse live logs on a host without collecting the logs and sending them to a SaaS Server. The user can select a rule, source, and some filters from the UI and the search is sent down to agent to obtain log files that match and bring them back, storing them in a temporary storage in the server. One use case for this feature is to allow users to browse a short time period of log files across multiple targets in a system to try to discover a source of a problem, especially when there is a rich topology mapping and dependency mapping of the customer's environment. This content can be used to help find related elements and show the logs together. This allows the users to see logs for all targets related to a given system for instance and see what happened across all targets in time sequence. In many cases, when there is a target failure, it may be a dependent target that is experiencing the problem, not the target that is failing.

The user may choose to start a new log browsing session in context of a system/group/individual target. If coming in from a target home page, the target home page context is to be retained. This means that the outer shell of the page still belongs to the target home page, and just the content panel will contain the browse UI functionality. This means the browse UI can be implemented to be modular to plug into other pages dynamically. In some embodiments, multiple row-content can be provided per entry to show additional details per row. This is one row at a time, or the user could decide to perform this for all rows. Sorting can be provided on the parsed fields, but in addition, can be used to see additional details per row (including the original log entry).

Search filters can be provided. For example, a search filter in the form of a date range can be provided, e.g., where the options are Most Recent, and Specific Date Range. With the Most Recent option, the user can enter some time and scale of Minutes or Hours. With the Specific Date Range, the user will enter a start and end time. With the date range option, Targets, Sources, and Filters can be specified. These allow the users to select what they want to see in this log browsing session. After the user has selected the targets, sources, and applied any filters, they can begin the browse session to initiate retrieval of the logs from various targets and ultimately have them shown on the interface.

Search queries can be implemented in any suitable manner. In some embodiments, natural language search processing is performed to implement search queries. The search can be performed across dependency graphs using the search processing. Various relationships can be queried in the data, such as "runs on", "used by", "uses", and "member of".

In some embodiments, the search query is a text expression (e.g., based on Lucene query language). Users can enter search query in the search box to search logs. The following are example of what could be included in the search query: (a) Terms; (b) Fields; (c) Term modifiers; (d) Wildcard searches; (e) Fuzzy searches; (d) Proximity searches; (f) Range searches; (g) Boosting a term; (h) Boolean operators; (i) Grouping; (j) Field grouping; (k) Escaping special characters.

A tabular view can be provided of the search findings. Some query refinement can be performed via table cells to allow users to add/remove some field-based conditions in the query text contained in the search box via UI actions. For example, when a user right-mouse clicks a field, a pop-up provides some options for him/her to add or remove a condition to filter the logs during the searches. This is convenient for users to modify the query text, and with this approach, users do not need to know the internal field names to be able to refine the query at field level.

There are numerous ways that can be provided to list fields for user to select/de-select them for display purpose in the search findings table. One example approach is based on static metadata, and another possible way is based on dynamic search results.

For list fields based on static metadata, a basic field shuttle is used to list all defined fields. Some example fields that can be defined by the log entry metadata include: (a) Log file; (b) Entry content; (c) Rule name; (d) Source name; (e) Parser name; (f) Source type; (g) Target type; (h) Target name. The values of these fields can be obtained from the agent with log entry (although source, parser, rule, target are all GUIDs/IDs) that will need to be looked up at display time.

For list fields based on dynamic search findings, the top n fields (e.g., 10) will be shown that would be suggested as making the most difference for that search. A "more fields" link will lead to a popup for users to select other fields. Users can see more information of those fields on the popup than form the View menu. When listing the fields, the system could use any suitable algorithm, for example, to assign a number to each field that is influenced by how many rows in the search results having non-null value, or how many different values there are across all search results for that field, etc.

Given so many dynamic fields available for users to select/de-select, it is desired for a user to be able to save the fields selection (field names and sizes). The system can store the last selected fields so when the user comes back to the page, he/she still gets the fields picked last time.

There may be a very large number (e.g., thousands) of log entries resulting from a search and it may not be possible for users to browse all of them to find the interested logs. For a particular search, users should be able to drill down to the details of the search findings with a few clicks. In some embodiments, features include clickable bar charts and table pagination. With these navigation features, plus customizable time range, users should be able to jump to some interested point quickly. Correspondingly, some embodiments provide for drilling up from details to higher levels so users can easily navigate to desired log entries via bar graphs. An example use case is: after users drill down a few levels they may want to drill up back to a previous level to go down from another bar. After users identify an interested log entry via some searches, they likely want to explore logs from a particular log source around the interested log entry, or explore logs from multiple log sources around the interested log entry in time-interleaved pattern. Some embodiments provide an option for users to browse forward/backward the logs around a specified log entry page by page. A graphical view can be provided of the search findings. This allows the user to pick fields to render the results graphically.

Some embodiments pertain to improved techniques to address log distributions, trends, and correlations. For search findings resulted from a particular search, distributions can be based on log counts to give users some high-level information about the logs. For each distribution type, the top n (e.g., 5 or 10) items are listed with number of found logs (where a "more . . . " link will lead to a popup with all other items listed). When users select a particular item, only logs corresponding to that item would be shown in the right table, so the action is equivalent to filtering the search findings with that item. Such information may be presented: (a) By target type; (b) By target, such as target owner and/or lifecycle status; (c) By log source; (d) By tag. Besides showing the search findings in the results table, the system can also provide options for users to switch between table view and the corresponding distribution chart view.

In some embodiments, results can be filtered by selecting distribution items. Users can filter the results table by selecting one or more distribution items. By default, all distribution items are selected and all log entries are listed in the results table. After selecting one or more distribution items, users can navigate the log entries via pagination. With one or more distribution items selected, when users click the search button for a new search, the selections of distribution items will be reset to be selected for all distribution items.

Some embodiments provide a feature to show search finding trends. Some embodiments provide a feature to show search finding correlations. Related to this feature, some embodiments provides launching links for users to navigate to search/view detailed logs when they perform correlation analysis among events, metrics, and infrastructure changes. Launching links could be provided, e.g., for users to navigate to an IT analytics product to analyze/view detailed events/metrics when they wish to see some bigger pictures related to the logs here.

Another feature in some embodiments pertains to process-time extended field definitions. Even with the same baseline log type, it is possible for individual log entries to contain inconsistent information from one log to the next. This can be handled in some embodiments by defining base fields common to the log type, and to then permit extended field definitions for the additional data in the log entries.

To explain, consider that a source definition defines log files to monitor. The log files are parsed into their base fields based on the log type definition. One can extract additional data that is not consistent across all log entries, e.g., as shown in **1300** of FIG. **13**. In this figure, the base fields that are parsed from the log entries are Month, Day, Hour, Minute, Second, Host, Service, Port (optional), and Message. The goal is to extract IP address and Port out of the second log entry. This goal may not be obtainable in certain implementations as part of the log type, e.g., since not every log entry has this structure. Here, the Message field for the second entry has the following content:

Accepted publickey for scmadm from xxx.xxx.1.1 port xyz ssh2

In some embodiment, a definition is made for an Extended Field Definition on the Message field using a format such as:

Accepted publickey for .* from {IP Address} port {Port} ssh2

For that log entry, two new field IP Address and Port will be parsed out and will be usable for reporting, searching, etc. This extraction happens as the data is being processed at collection time.

According to some embodiments, the processing for implementing process-time extended field definitions comprises: identifying one or more log files to monitor, wherein some of the entries in the one or more log files may include additional data that does not exist in other entries or is inconsistent with entries in the other entries, such as an additional IP address field in one entry that does not appear in another entry; identifying a source definition for one or more log files to monitor; parsing the one or more log files into a plurality of base fields using the source definition; defining one or more extended fields for the one or more log files; and extracting the one or more extended fields from the one or more log files.

Therefore, some embodiments permit the user to add extended field definitions. These are defined patterns that are seen within a field. A user could perform a create-like on a source and then the source and all extensions will become a new user-created source. The extended field definition defines new fields to create based on the content in a given file field. In some embodiments, the extended field definitions (and tagging) can be applied retroactively. This allows past log data to be processed with after-defined field definitions and tags.

FIG. **14** shows some example field definitions **1302**. For the first case in the table, the user is specifying to look at the "Message" file field that comes from the log entry and is parsed by the file parser. This Message field will have text in it, but the user has identified that they want to capture the SIGNALNAME part of the message as a new field for this specific message. This new field (SIGNALNAME) can now become viewable in the captured log entries, viewable in the Log Browser, and can also be stored as part of a metric if a rule is created to do so. The extended field definition uses the entire contents of the Message in this example. The user could bind either side of their expression with a wildcard pattern. For instance, the definition could have been simply "sending a {SIGNALNAME}". The text that is shown is known to be static text that never changes for this log message. The use of [0-9]* in the expression means that any number of numeric characters can be located here, but they will just be ignored (since there is no field name associated to name this field. The text that comes after the string "sending a" will get assigned to the variable SIGNALNAME.

The last entry is another example where the user has defined two new fields and in the first field, they have also defined the way to get this content using a regular expression. Here, there are some characters containing a-z, A-Z, 0-9 or a hyphen before a period '.'. Everything that matches that expression should be added to a new extended field called the HOSTNAME. Anything after the first period will be put into a new extended field called DOMAINNAME. The HOST field which came from the file parser will still have all of the content, but this extended field definition is telling our feature to add two NEW fields in addition to the HOST field (HOSTNAME and DOMAINNAME).

All extended field definitions where a new field is defined using the { } delimiters uses a parse expression. However in this example, except the HOSTNAME field in the last example, there is none shown. This is because in some embodiments, there is a default known regular expression pattern of (.)* which means any number of character. This expression is implicitly used if the user does not provide a regular expression. If there is static text, the system will take any characters between the two pieces of static text. If there is no static text or characters after a field expression, it is assumed that every character to the end of the file field is part of the new extended field's value (like DOMAINNAME in the last example and CONTENT_LENGTH_LIMIT in the third example.) This could lead to some issues if there were variants of this log entry that have additional text sometimes. The way to solve this is to also define the parse regular expression for each field and not rely on the default implicit (.)*.

Some embodiments provide the ability to define regular expressions and save them with a name. For instance, the regular expression for hostname used above is [a-zA-Z0-9\−]+.

One example of a saved regular expression may be:

IP_Address Regular Expression=>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}

When referencing this saved regular expression in the extended field definition, the last entry in the table above may look like this instead:

{HOSTNAME:@IP_Address}.{DOMAINNAME}

The new fields that will be created are HOSTNAME and DOMAINNAME. The referenced regular expression that was created and saved is called IP_Address. When the system performs the processing on the agent, it will replace the referenced regular expression "@IP_address" with the regular expression string:

"\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"

Extended expression definitions can be evaluated at the agent (e.g., using a Perl parsing engine) directly with minor changes to the input string from the user.

In some embodiments, field reference definitions can be provided. This provides a feature where users can provide a lookup table of a SQL query to transform a field which may have a not-easily-readable value into more human readable content. Three example use cases highlight this need: (a) In a log entry, there may be an error code field (either a core field or an extended field) that simply has a number, where the user can provide a lookup reference so that the system adds another new field to store the textual description of what this error code means; (b) In a log entry, there may be a field (either a core file field or an extended field) that has the GUID of a target, and the system can provide a lookup using a SQL query to a target table that will create another new field that stores the display name of the target; (c) IP to hostname lookup may also be performed as a common use case, where in a log, there may be IP addresses for clients, where the IP addresses are used to look up hostnames.

As noted above, log types (also referred to herein to include "Parsers" in some cases in this document) may also be defined to parse the log data. One example log type pertains to the "Log Parser", which is the parser that can be used to parse the core fields of the source. Another example log type pertains to a "Saved Regular Expressions", which can be used when defining extended field definitions. For example, a hostname can be defined via a regular expression as "[a-zA-Z0-9\−]+". This regular expression can be saved with a name and then used a later time when creating extended field definitions.

A log parser is a meta-data definition of how to read a log file and extract the content into fields. Every log file can be described by a single parser to break each log entry into its base fields. The log type may correspond to a parse expression field, such as for example, a Perl regular expression for parsing a file. When defining a log parser, the author identifies the fields that will always exist in the log file. In this case, the following are the fields that exist in every entry of the above log file:

Some fields may be very complex, meaning that the field will actually contain additionally structured content for some log entries but not for others. These may not be handled by the log file parser in some embodiments because it is not consistent in every line. Instead, when defining a source, extended fields can be defined to break this field into more fields to handle these cases.

Profiles can be implemented for various constructs in the system, such as parsers, rules, and sources. The profiles capture differences between different usages and/or versions of data items and products for users. For example, a source profile can be created that accounts for different versions of a user's products that are monitored, e.g., where a source profile changes the source definition between version 1 and version 2 of a database being monitored. Rule profiles may be used to account for differences in rules to be applied. As another example, parser profiles can be provided to adjust parsing functionality, e.g., due to difference in date formats between logs from different geographic locations. Different regular expressions can be provided for the different parser profiles.

With regards to a log entry delimiter, log files can have content that is always known to be one row per entry (syslog), or can have content that can span multiple lines (Java Log 4j format). The Log Entry Delimiter input lets the user specify to always parse this log file as one row per entry, or to provide a header parse expression that tells us how to find each new entry. The entry start expression will typically be the same as the first few sections of the parse expression. The system uses this expression to detect when a new entry is seen versus seeing the continuation of the previous entry. For this example, the entry start expression may be:

```
([A-Z]{1}[a-z]{2})\s([0-9]{1,2})\s([0-9]{1,2}):([0-9]{2}):([0-9]{2})
```

This expression looks for a strict month, day, hour, minute, second structure. If that exact sequence of characters is seen, this "line" is treated as the beginning of a new entry.

In some embodiments, a table is maintained corresponding to parsed fields, and which starts empty (no rows) as the parse expression is empty. As users are creating the parse expression, the fields being defined are added to this table. This can be implemented by monitoring the text entered in this field and when a ')' is added, a function is called to determine how many fields have been defined. The system can ignore some cases of (and), e.g., when they are escaped or when they are used with control characters.

For instance, consider the following parsing language:

```
([a-z] {2})\s([a-z0-9]+)
```

In this example, there are two pairs of ( ) which means there are two fields defined. The content inside is how to find the field from the log entry—The UI for this create parser page does not care about what is inside the parenthesis. This is evaluated and used on the agent only. The content outside of the (and) are just static text that helps parse the line (this UI also does not care about this). For creating the right number of fields in the table, the approach counts the number of ( ) pairs in the parse expression. For each field that is parsed out by the parse expression, the user provides a field name based on one of the existing common fields.

Automated Classifications

Some embodiments of the invention provide an approach to perform machine learning-based classification of logs. This approach is used to group logs automatically using a machine learning infrastructure. The general idea is that given the data from within logs to be analyzed, one can automatically group the logs into appropriate categories (e.g., to automatically identify the log type for the log).

FIG. **15** shows an architecture for performing machine learning-based classification of logs according to some embodiments of the invention. A known set of logs is acquired to form a baseline set of data for log groupings, where the known set of logs are used to form learning models **1505***a* and **1505***b*. The learning models **1505***a* and **1505***b* identify characteristics of known log types, so that unknown logs **1501***a-c* that are received can be matched against those characteristics to classify those logs within one of the known log types.

Logs **1501***a-c* are gathered from one or more computer readable mediums **1503** within a customer environment. The logs **1501***a-c* may undergo filtering by a filter **1502**. Within the filter **1502**, known patterns of content are removed from the log data prior to classification. For example, it may be desirable to remove certain types of log content exceptions (such as Java exceptions) from the log data prior to classification. Therefore, one or more content patterns that correspond to a Java exception may be used to filter our any Java exceptions from the logs **1501***a-c.*

Within a machine learning infrastructure **1504**, a set of classifiers **1504***a* and **1504***b* use the learning models **1505***a* and **1505***b*, respectively, to classify the logs **1501***a-c*. In some embodiments of the invention, multiple classifiers **1504***a* and **1504***b* are employed to classify the logs **1501***a-c*. The idea is that each classifier operates with a different set of parameters and assumptions from the other classifiers. By using multiple classifiers, this ensures that a corner case that may negatively affect the accuracy of one classifier would not completely destroy the ability to confidently generate a classification, since one or more other classifiers not significantly affected by the corner case would also be operating to classify the data. Careful selection of the two classifiers should allow the classification architecture **1504** to avoid misclassifications due to such data skews by all the classifiers, even if one of the classifiers may be affected.

In some embodiments, a first classifier, referred to herein as a "distribution classifier", may operate based upon vectors generated by the distribution/frequency of characters within the log. A second classifier, referred to herein as a "token classifier", may operate based upon vectors generated by identification of certain tokens within the log. A third type of classifier may operate by matching log content against one or more regular expressions, where a given log type of associated with one or more regular expressions. For example, a certain log type may always include the following in each log entry: "Name=[alphabetic name]". In this case, the regular expression "Name\=[a-z]+" may be used to match against any logs that correspond to this log type. A fourth type of classifier may operate by checking for a pattern signature, where a given pattern signature corresponds to a given log type. The pattern signature may be expressed, for example, by identifying both fixed and variables portions, and checking whether the log data matches against that signature. A simple example of a signature could be "Name=[variable portion] Date=[Variable portion]", where the "Name=" and "Date=" sections of this line are fixed portions, and classification operates by checking for this signature pattern within an unknown log.

While certain example classifiers are described above, it is noted that other classifier types may also be employed within the scope of the invention. In addition, while FIG. **15** only shows two classifiers used in conjunction with one another, it is noted that the inventive concept described herein may be applied with any number of classifiers, and is expressly not limited just to two classifiers.

Assume that classifier **1504***a* is a distribution classifier and classifier **1504***b* is a token classifier. Each unknown log within logs **1501***a-c* is processed to generate a vector for that log, e.g., by identify term frequencies within the log for a distribution classifier and token-based vectors for a token classifier. The vector(s) are then compared against the data within the learning models **1505***a* and **1505***b* for the known set of logs. A similarity comparison is then performed to classify the log and to generate classification results **1510**.

FIG. **16** shows a high level flowchart of an approach to implement machine learning-based classification of logs according to some embodiments of the invention. At **1602**, the process begins with a training phase to generate learning models for the classification process. As described in more detail below, a supervised training process is performed to implement the training phase.

Next, at **1604**, the log data to be classified is gathered. As described above, one or more log gatherers may be located within the client environment to gather the log data. The log data is gathered and passed to a log analytics system that includes the machine learning infrastructure for classifying logs.

At **1606**, the gathered log data is analyzed relative to the learning models, where content within the log data is vectorized to generate one or more vectors. The vectors are then compared against the data within the learning models. In some embodiments, multiple classifiers may operate against the vectorized data. Based upon this analysis, at **1608**, classification is performed to classify the log as a recommended log type (or multiple recommended log types).

Once a log has been properly classified, that log can now be parsed using the appropriate log parser that has been constructed for that log type. The log parser may include a set of regular expressions specifically constructed for a given log type to extract a designated set of fields and values from the log. Therefore, failure to identify the correct log type may cause a failure of the log parsing process, especially if the wrong log parser is used to parse the log. With the present approach, a high level of confidence can be gained for the proper identification of the correct log type for an unknown log. This serves to more efficiently and effectively implement the parsing stage of log processing, since the appropriate log parser can then be selected to parse the log.

FIG. **17** illustrates a flowchart of an approach to implement the learning phase according to some embodiments of the invention. At **1702**, a set of logs is identified that correspond to known log types. This set of known logs forms the basis of the training data. The set of known logs may comprise an initial set of training material, or may be provided as follow-up training materials from a feedback process where previous incorrectly-classified logs are identified and placed within the training materials to improve the accuracy of the learning models.

At **1704**, the training data is organized by the data's known log types. In some embodiments, a directory structure is employed to organize the known logs, where each log type corresponds to a sub-directory having the logs that are known to be of that log type. Any number of these directory structures may be formed, to hold data for a respective number of log types for which the training phase is intended to process.

At **1706**, the log data is vectorized on a log-type basis. Each set of logs within a sub-directory for a given log type is transformed into a vector and plotted onto a coordinate space. From those vectors, clusters of vectors can be formed, where at **1708**, one or more centroids identified for the cluster(s). In this manner, a centroid can be identified for each log type that is currently recognized by the system.

FIG. **18** illustrates one possible approach for organizing the known set of log data within the log analytics system. A top level directory holds all of the sub-directories for each log type. Here, the top level directory includes sub-directories for log type 1, log type 2, . . . log type n. Each sub-directory includes the log files within the training set that are known to correspond to the log type for that sub-directory. The sub-directory for Log type 1 includes log file_1$_{type\ 1}$, log_file_2$_{type\ 1}$, . . . log_file_n$_{type\ 1}$ that are all within the training set and are known to correspond to log type 1. Similarly, the sub-directory for Log type 2 includes log_file_1$_{type\ 2}$, log_file_2$_{type\ 2}$, . . . log_file_n$_{type\ 2}$ that are all within the training set and are known to correspond to log type 1.

FIGS. **19-1** through **19-5** illustrate the process for generating the learning model for the different log types. FIG. **19-1** shows three example logs Log 1, Log 2, and Log 3. It is assumed that all three logs are within a training set and are known to be of the same log type. Log 1 includes the following content: "Name=Bob Date=May 1 URL=www.xyz.com/abcdefghijk/lmnopq". Log 1 includes the following content: "Name=Joe Date=April 5 URL=www.123.com/4567893409344/dfjgoms". Log 3 includes the following content: "Name=Sam Date=June 25 URL=www.abc.com/rtiosprmskfl/eroskuf".

The training process begins by converting each of these logs into a vector value, and then plotting the vector values within a coordinate space. FIG. **19-2** illustrates this process for Log 1. Here, the log is converted into a first type of vector (distribution vector **1904***a*), where the distribution/frequency of each character within the log is considered to generate the vector. Vector **1904***a* is then plotted as a point **1911***a* within coordinate space **1910**. In a similar way, the log is converted into a second type of vector (token vector **1906***a*) where the tokens within the log (or at least the top n tokens within the log) are used to generate the token vector **1906***a*. Vector **1906***a* is then plotted as a point **1913***a* within coordinate space **1912**.

Each of the other logs undergoes this same process. FIG. **19-3** illustrates processing for Log 2, where the log is converted into a first distribution vector **1904***b* and a second token vector **1906***b*. The distribution vector **1904***b* is plotted as a point **1911***b* within the coordinate space **1910**, and the token vector **1906***b* is plotted as a point **1913***b* within the coordinate space **1912**. Similarly, FIG. **19-4** illustrates processing for Log 3, where the log is converted into a first distribution vector **1904***c* and a second token vector **1906***c*. The distribution vector **1904***c* is plotted as a point **1911***c* within the coordinate space **1910**, and the token vector **1906***c* is plotted as a point **1913***c* within the coordinate space **1912**.

Next, as shown in FIG. **19-5**, clustering is performed to identify clusters of points within the plotted vector points. A similarity radius can be established to identify the vectors that cluster together within each set of plots. Within the coordinate space **1910** for the distribution vectors, a similarity radius **1917** has been established which groups points **1911***a*, **1911***b*, and **1911***c* into the same cluster. For the coordinate space **1912** that corresponds to the token vectors, a similarity radius **1919** has been established which groups points **1913***a*, **1913***b*, and **1913***c* into the same cluster.

Centroids can then be identified for each cluster. Any suitable approach can be taken to identify the centroids for the clusters. For example, the following equation may be used to identify the center of a cluster: Center=$\Sigma W_i V_i$, where i refers to the identifier for a given vector V within the cluster, and W refers to a weight that is assigned to that vector. In some embodiments, the weight W is determined by dividing the number 1 by the total number of characters within that log. This approach to weighting serves to provide normalization for the different logs with respect to the number of characters that may exist within any given log.

In the example of FIG. **19-5**, application of the above formula results in identification of centroid **1921** for the cluster of distribution vectors, and centroid **1923** for the cluster of token vectors. This set of data forms the models that can be used to classify a set of unknown logs.

FIG. **20** shows a flowchart of an approach to implement classification according to some embodiments of the invention. At **2002**, identification is made of the log to be analyzed. This action retrieves one of the unknown logs for processing, e.g., a log that was gathered from a client location and imported into a log analytics system.

Next, at **2004**, vector data is generated for the log. This is implemented, for example for the distribution classifier, by performing feature extraction to identify the frequency of terms within the log data, and to then construct term-frequency vectors that correspond to the log data. For a token classifier, identification of tokens within the log is performed to generate a token vector for the log.

At **2006**, a similarly comparison is performed for the generated vector data. In some embodiments, known log data may have been used to construct comparison data, e.g., by vectorizing the sample data and constructing clusters for the sample data using an appropriate clustering algorithm. The similarly comparison is performed by comparing the vector data for the log under analysis to the vector data for the known samples (e.g., against the centroid of the cluster for the known samples).

At **2008**, categorization of the log can be performed using the outputs of the comparison process. Distance thresholds may be established to determine whether the log under analysis is similar enough to be classified within the category associated with one or more of the known samples. For example, analysis may be performed to determine the distance(s) between the vector for the log under analysis and the various centroids that have been identified for the known log types, where a probability value is determined for some or all of the known log types.

FIGS. **21-1** through **21-11** illustrate this classification process. FIG. **21-1** shows both a distribution model and a token model. The distribution model includes a first centroid (Dist_Centroid$_{Log\ 1}$) corresponding to a first log type, a second centroid (Dist_Centroid$_{Log\ 2}$) corresponding to a second log type, and a third centroid (Dist_Centroid$_{Log\ 3}$) corresponding to a third log type. Similarly, the token model includes a first centroid (Token_Centroid$_{Log\ 1}$) corresponding to a first log type, a second centroid (Token_Centroid$_{Log\ 2}$) corresponding to a second log type, and a third centroid (Token _Centroid$_{Log\ 3}$) corresponding to a third log type. A distribution classifier **2104***a* operates to classify logs according to the distribution model and the token classifier **2104***b* operates to classify logs according to the token model.

FIG. **21-2** illustrates a log data **2110** being received for classification. Assume that the log type of log data **2110** is currently unknown. FIG. **21-3** shows log data **2110** being directed to the distribution classifier for processing. As illustrated in FIG. **21-4**, log data **2110** is transformed into a distribution vector **2105***a*, e.g., based upon the distribution and/or frequency of characters within log data **2110**. Vector **2105***a* is then plotted within the coordinate space of the distribution model, as shown in FIG. **21-5**.

At this point, distances are calculated between vector **2105***a* and the centroids for the known log types. As shown in FIG. **21-6**, Distance$_{log\ 1}$ identifies the distance between vector **2105***a* and the centroid Dist_Centroid$_{Log\ 1}$ corresponding to a first log type, Distance$_{log\ 2}$ identifies the distance between vector **2105***a* and centroid Dist_Centroid$_{Log\ 2}$ corresponding to a second log type, and Distance$_{log\ 3}$ identifies the distance between vector **2105***a* and the centroid Dist_Centroid$_{Log\ 3}$ corresponding to the third log type. These distances will later be used to create classification recommendations for the log data **2110** relative to each of the log types 1, 2 and/or 3.

FIG. **21-7** shows log data **2110** being directed to the token classifier for processing. As illustrated in FIG. **21-8**, log data **2110** is transformed into a token vector **2105***b*, e.g., based upon tokens identified within log data **2110**. Vector **2105***b* is then plotted within the coordinate space of the token model. Distances are calculated between vector **2105***b* and the centroids within the token model for the known log types. As shown in FIG. **21-9**, Distance$_{log\ 1}$ identifies the distance between vector **2105***a* and the centroid Token_Centroid$_{Log\ 1}$ corresponding to a first log type, Distance$_{log\ 2}$ identifies the distance between vector **2105***a* and centroid Token_Centroid$_{Log\ 2}$ corresponding to a second log type, and Distance$_{log\ 3}$ identifies the distance between vector **2105***a* and the centroid Token_Centroid$_{Log\ 3}$ corresponding to the third log type.

At this point, as shown in FIG. **21-10**, calculation are performed to identify the appropriate log type that should be recommended for unknown log data **2110**. The distance from the log vector to the centroids for each of the known log types is analyzed, where the closer the distance from the vector to a given centroid, the more likely it is that the log should be classified as the log type associated with that centroid. Similarly, the greater the distance from the vector to a given centroid, the less likely that log should be classified as the log type associated with the centroid.

According to some embodiments, the results from both (multiple) classifiers are considered to identify the appropriate log type for the log. Weighting may be applied to associate the appropriate weight for each type of classifier to the final results. For example, assume that the distribution classifier is intended to contribute to 20% of the final results, whereas the token classifier is intended to contribute 80% to the final results. In this situation, the weight W_{Dist} for the distribution classifier would be set to 0.2 and the weight W_{Token} for the token classifier would be set to 0.8.

FIG. **21-11** illustrates one possible approach to display classification results within a user interface on a display device. In this figure, each of the different log types are presented, along with a percentage probability that the log should be classified as that log type. In some embodiments, the list is sorted, and only the top n log types with the highest probability percentages are displayed in the interface. In an alternate embodiment, instead of displaying detailed percentage probabilities, one (or more) recommended classifications are provided for only those log types which meet a threshold level of similarity (e.g., by establishing a similarity threshold radius when comparing the log vector to log type centroids).

During the learning phase, processing can be performed to identify an optimal model for log classification. Recall that a similarity radius is established to identify clusters within a training set of data, where vectors that fall within the scope of the similarity radius can be clustered together. However, vectors that fall outside the scope of the similarity radius may fall within the scope of another cluster. It is quite possible for a given log type to correspond to multiple clusters, and hence multiple centroids within a learning model. In fact, if the vectors are spread wide enough, it is possible in the most extreme case for each sample log for a log type in the training set to correspond to its own centroid. The issue is that the more clusters that exist for a given log type, the more work may be needed during the classification process to compare an unknown log against each of the centroids. Therefore, it is desirable to identify the least number of centroids that nevertheless will allow each and every sample log within the training set to classify within the radius of at least one of the minimal number of centroids.

FIG. **22** shows a flowchart of an approach to perform this type of processing. At **2202**, the process begins with either a maximum number of centroids (and works its way to a smaller and more optimal set of centroids) or a minimum number of centroids (and works its way to more centroids if needed). With the maximum number, each centroid is essentially centered at one of the sample logs in the training set for the log type being trained. At this point, at **2204**, a determination is made of the coverage of the sample logs that fall within the similarity radius of the centroid(s) as well as the extent of the similarity radius. In particular, if there are any sample logs at all that do not fall within the scope of one of the clusters, then the numbers of clusters must be adjusted to create a new cluster and/or the similarity radius needs to be adjusted to account for the coverage error. In addition, if too many clusters have been identified, then the overlap in coverage by the clusters can be determined at this point.

At **2206**, a determination is made whether any adjustments need to be made. For example, a determination made be made, at **2208**, to adjust the number of centroids. Depending upon the direction of the processing (e.g., starting from max number of centroids or min number of centroids), an action may be taken to either increase to decrease the number of centroids. In addition, at **2210**, the determination may be made to adjust the similarity radius for the clustering. A maximization function may be performed to identify the optimal set of centroids/radiuses. In addition, a binary search may be performed to identify one or more optimal solutions.

Under both approaches, the process returns back to **2204** to determine whether any additional adjustments are needed. If the current configuration of centroids/radius is acceptable, then at **2212**, the model is outputted.

FIG. **23** illustrates the situation where the original similarity radius **2312***a* was inadequate to correspond to the vectors to be clustered, e.g., where the un-clustered vectors are known to be for the exact same log type as the vectors that are actually in the cluster and hence failure to include the un-clustered vectors constitutes an error in coverage for the cluster. In some situations to correct this problem, the radius can be expanded into a modified similarity radius **2312***b*. This modified radius **2312***b* now correctly clusters all vectors for the log type without any classification errors.

FIG. **24** illustrates the situation where the original set of centroids is updated to reflect a new set of centroids. Here, the original analysis identified only a single centroid **2414***a* for cluster **2412***a*. The issue is that there are additional vectors that are supposed to be the same log type as the vectors for cluster **2412***a*, but do not classify that way if only the single centroid **2414***a* is in the model. In this situation, a new centroid **2414***b* can be identified for a second cluster **2412***b* of vectors for that log type. Here, the model include both centroids **2414***a* and **2414***b* for the log type, and therefore unknown logs are classified relative to both centroids during the classification process.

Various types of post-modeling testing may be performed to check the accuracy of the models. One possible test is to identify another set of known logs that are known to be of the same log type that was modelled. These additional logs are run through a classifier relative to the models to determine if they match the correct classifications, within an acceptable threshold. If not, then adjustments may be made to correct the possible issues, e.g., by adding additional data from the new set of data to the training set to correct the models.

An additional optimization that can be performed is to pre-process the log data to improve the accuracy of the classifications. To explain, consider the log data shown in FIG. **26-1**. Here, both Log 1 and Log 2 include portions that are very similar (e.g., "Name= . . . " and "Date= . . . "), but also include very lengthy portions that are very dissimilar (e.g., the URL portions). In this situation, when converting the logs to vectors, the significantly different URL portions may overwhelm the parts that are similar, creating excessive mismatches between the vectors for logs that should be classified as the same type. This issue may be partially addressed by using the token vectorization approach, but log type/content outliers may nonetheless still create accuracy issues.

FIG. **25** shows a flowchart of an approach that can be taken to address this type of problem. At **2502**, pre-processing is performed to analyze the contents of the log data. In particular, at **2504**, common portions and variable portions of the log are identified. The teachings of U.S. application Ser. No. 14/863,136, filed on Sep. 24, 2015, which is hereby incorporated by reference in its entirety, can be used in conjunction with this embodiment, to identify the constant parts and the variable parts of the log.

At **2506**, the variable parts are removed from consideration for the analysis process. In particular, only the constant parts are considered when generating vectors from the log data. Thereafter, at **2508**, classifications models are generated using only the constant portions of the logs.

FIGS. **26-1** through **26-3** illustrate this process. As noted above, FIG. **26-1** shows two logs, where both Log 1 and Log 2 include portions that are very similar (e.g., "Name= . . . " and "Date= . . . "), but also include very lengthy portions that are very dissimilar (e.g., the URL portions). As illustrated in FIG. **26-2**, the common portions and the variable portions are identified within the log data. For example, the "Name=", "Date=", and "URL=" portions are common between all of the logs for this log type. In contrast, the "Bob", "Joe", "May 1", "April 5", and URL portions vary between the two logs.

Therefore, the model generation process may operate only against the common portions. As shown in FIG. **26-3**, the variable portions may be removed from consideration, leaving only the common portions to be vectorized for model generation. During the classification process, the unknown logs may undergo vectorization in their entirety, or have variable portions removed in a pre-processing step.

In some cases, the variable portions do provide useful patterns that may be important for classification purposes. For example, consider again the logs shown in FIG. **26-1**. The variable portions after "Date=" includes "May 1" and "April 5". Even though these portions vary between the two logs, they are just not random sets of characters, but instead may have meaningful contribution given that they are recognizable as date fields. As such, it may be advantageous in certain circumstances to retain these variable fields during the model generation and classification process.

FIG. **27** shows a flowchart of an approach that can be taken to implement this aspect of some embodiments of the invention. At **2702**, pre-processing is performed to analyze the contents of the log data. At **2704**, common portions and variable portions of the log are identified.

In addition, at **2706**, "field rule" types may be identified from the variable portions of the log. This type corresponds to any sequence of characters that is identified based upon a rule definition, and may correlate to complex combinations of any numbers of characters, integers, or symbols. A regular expression may be used to express the rule for this type. The teachings of U.S. application Ser. No. 15/089,180, filed on even date herewith, which is hereby incorporated by reference in its entirety, can be used in conjunction with this embodiment, to identify the field rule types within a log.

At **2708**, the variable parts are then removed from consideration for the analysis process. This leaves both the constant portions and the field rule portions to be considered when generating vectors from the log data. Thereafter, at **2710**, classifications models are generated using only the constant portions of the logs.

FIGS. **28-1** through **28-4** illustrate this process. FIG. **28-1** shows two logs, where both Log 1 and Log 2 include portions that are very similar (e.g., "Name= . . . " and "Date= . . . "), but also include very lengthy portions that are very dissimilar (e.g., the URL portions). As illustrated in FIG. **28-2**, the common portions and the variable portions are identified within the log data. For example, the "Name=", "date=", and "URL=" portions are common between all of the logs for this log type. In contrast, the "Bob", "Joe", "May 1", "April 5", and URL portions vary between the two logs.

Here, the variable portions "May 1" and "April 5" can be recognized as date fields. Therefore, as shown in FIG. **28-3**, these variable portions are identified as date types within the log.

Thereafter, the model generation process may operate only against the common portions and the date type portions. As shown in FIG. **28-4**, the variable portions may be removed from consideration, leaving the common portions ad the date portions to be vectorized for model generation. During the classification process, the unknown logs may undergo vectorization in their entirety, or have variable portions removed in a pre-processing step.

Therefore, what has been described is an improved system, method, and computer program product for implementing a log analytics method and system that can configure, collect, and analyze log records in an efficient manner. In particular, machine learning-based classification can be performed to classify logs. This approach is used to group logs automatically using a machine learning infrastructure.

System Architecture Overview

FIG. **29** is a block diagram of an illustrative computing system **1400** suitable for implementing an embodiment of the present invention. Computer system **1400** includes a bus **1406** or other communication mechanism for communicating information, which interconnects subsystems and devices, such as processor **1407**, system memory **1408** (e.g., RAM), static storage device **1409** (e.g., ROM), disk drive **1410** (e.g., magnetic or optical), communication interface **1414** (e.g., modem or Ethernet card), display **1411** (e.g., CRT or LCD), input device **1412** (e.g., keyboard), and cursor control.

According to one embodiment of the invention, computer system **1400** performs specific operations by processor **1407** executing one or more sequences of one or more instructions contained in system memory **1408**. Such instructions may be read into system memory **1408** from another computer readable/usable medium, such as static storage device **1409** or disk drive **1410**. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and/or software. In one embodiment, the term "logic" shall mean any combination of software or hardware that is used to implement all or part of the invention.

The term "computer readable medium" or "computer usable medium" as used herein refers to any medium that participates in providing instructions to processor **1407** for execution. Such a medium may take many forms, including but not limited to, non-volatile media and volatile media. Non-volatile media includes, for example, optical or magnetic disks, such as disk drive **1410**. Volatile media includes dynamic memory, such as system memory **1408**.

Common forms of computer readable media includes, for example, floppy disk, flexible disk, hard disk, magnetic tape, any other magnetic medium, CD-ROM, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, RAM, PROM, EPROM, FLASH-EPROM, any other memory chip or cartridge, cloud-based storage, or any other medium from which a computer can read.

In an embodiment of the invention, execution of the sequences of instructions to practice the invention is performed by a single computer system **1400**. According to other embodiments of the invention, two or more computer systems **1400** coupled by communication link **1415** (e.g., LAN, PTSN, or wireless network) may perform the sequence of instructions required to practice the invention in coordination with one another.

Computer system **1400** may transmit and receive messages, data, and instructions, including program, i.e., application code, through communication link **1415** and communication interface **1414**. Received program code may be executed by processor **1407** as it is received, and/or stored in disk drive **1410**, or other non-volatile storage for later execution. Data may be accessed from a database **1432** that is maintained in a storage device **1431**, which is accessed using data interface **1433**.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. For example, the above-described process flows are described with reference to a particular ordering of process actions. However, the ordering of many of the described process actions may be changed without affecting the scope or operation of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense. In addition, an illustrated embodiment need not have all the aspects or advantages shown. An aspect or an advantage described in conjunction with a particular embodiment is not necessarily limited to that embodiment and can be practiced in any other embodiments even if not so illustrated. Also, reference throughout this specification to "some embodiments" or "other embodiments" means that a particular feature, structure, material, or characteristic described in connection with the embodiments is included in at least one embodiment. Thus, the appearances of the phrase "in some embodiment" or "in other embodiments" in various places throughout this specification are not necessarily referring to the same embodiment or embodiments.

What is claimed is:

1. A method comprising:

storing, by a log analytics system, a plurality of log parsers associated, respectively, with a plurality of log types;

obtaining, by the log analytics system, log data from a log;

identifying within the log data: (a) a first set of field names that are common to at least one log type of a plurality of log types, (b) a first set of variable portions comprising field values of fields represented by the first set of field names, and (c) a first set of field rule portions, wherein at least one of the first set of field names is defined based on log entry metadata;

generating filtered log data by removing the first set of variable portions from the log data, wherein the filtered log data comprises the first set of field names and does not include the field values of the fields represented by the first set of field names;

generating vectors that are (a) based on the first set of field names in the filtered log data and (b) not based on the removed first set of variable portions comprising the field values;

obtaining a final classification result that classifies the log as being of a particular type at least by applying, by the log analytics system, the vectors based on the first set of field names to one or more classifiers; and

based on the final classification result: parsing the log, by the log analytics system, using a log parser associated with the particular log type;

wherein the first set of field rule portions is retained in the filtered log data subsequent to removing the first set of variable portions;

wherein the method is performed by at least one device comprising a processor.

2. The method of claim **1**, further comprising:

applying, by the log analytics system, the filtered log data to a distribution classifier to obtain a first classification result, wherein the distribution classifier classifies the filtered log data using a distribution model comprising a first plurality of centroids that are associated, respectively, with the plurality of log types, wherein applying the filtered log data to the distribution classifier comprises (a) generating a distribution vector based on one or more frequencies of one or more characters within the filtered log data, and (b) generating the first classification result based on a first distance between the distribution vector and a first centroid in the first plurality of centroids;

applying, by the log analytics system, the filtered log data to a token classifier to obtain a second classification result, wherein the token classifier classifies the filtered log data using a token model comprising a second plurality of centroids that are associated, respectively, with the plurality of log types;

assigning a first weighting to the first classification result to obtain a first weighted classification result corresponding to the distribution classifier;

assigning a second weighting to the second classification result to obtain a second weighted classification result corresponding to the token classifier; and

combining at least (a) the first weighted classification result corresponding to the distribution classifier and (b) the second weighted classification result corresponding to the token classifier to obtain the final classification result.

3. The method of claim **2**, wherein:

applying the filtered log data to the token classifier comprises (a) generating a token vector based on one or more tokens within the filtered log data, and (b) generating the second classification result based on a second distance between the token vector and a second centroid in the second plurality of centroids.

4. The method of claim **2**, further comprising:

applying, by the log analytics system, the filtered log data to a regular expression classifier to obtain a third classification result; and

assigning a third weighting to the third classification result to obtain a third weighted classification result corresponding to the regular expression classifier;

wherein to obtain the final classification result, the log analytics system further combines the third weighted classification result corresponding to the regular expression classifier with the first weighted classification result corresponding to the distribution classifier and the second weighted classification result corresponding to the token classifier.

5. The method of claim **2**, further comprising:

applying, by the log analytics system, the filtered log data to a pattern signature classifier to obtain a fourth classification result; and

assigning a fourth weighting to the fourth classification result to obtain a fourth weighted classification result corresponding to the pattern signature classifier;

wherein to obtain the final classification result, the log analytics system further combines the fourth weighted classification result corresponding to the pattern signature classifier with the first weighted classification result corresponding to the distribution classifier and the second weighted classification result corresponding to the token classifier.

6. The method of claim **1**, further comprising:

determining a set of highest probability candidate log types for the log, the set of highest probability candidate log types comprising the final classification result and at least one other candidate classification result based on the filtered log data; and

generating a user interface comprising the set of highest probability candidate log types.

7. The method of claim **1**, wherein:

the log data comprises one or more log entries;

obtaining the log data from the log comprises receiving the one or more log entries from one or more log gatherers located in a distributed host environment.

8. The method of claim **2**, wherein:

based on the first weighting and the second weighting, the first classification result contributes a first percentage toward the final classification result and the second classification result contributes a second percentage toward the final classification result.

9. The method of claim **1**, the operations further comprise:

identifying (c) a first set of field rule portions in the log data, wherein the field rule portions are retained in the filtered log data subsequent to removing the first set of variable portions.

10. The method of claim **2**, wherein applying the filtered log data to the distribution classifier comprises applying a distribution of characters within the filtered log data to the distribution classifier.

11. The method of claim **2**, wherein applying the filtered log data to the token classifier comprises:

identifying one or more tokens within the filtered log data; and applying the one or more tokens within the filtered log data to the token classifier.

12. A non-transitory computer readable medium comprising instructions which, when executed by one or more hardware processors, causes performance for operations comprising:

storing, by a log analytics system, a plurality of log parsers associated, respectively, with a plurality of log types;

obtaining, by the log analytics system, log data from a log;

identifying within the log data: (a) a first set of field names that are common to at least one log type of a plurality of log types, (b) a first set of variable portions comprising field values of fields represented by the first set of field names, and (c) a first set of field rule portions, wherein at least one of the first set of field names is defined based on log entry metadata;

generating filtered log data by removing the first set of variable portions from the log data, wherein the filtered log data comprises the first set of field names and does not include the field values of the fields represented by the first set of field names;

generating vectors that are (a) based on the first set of field names in the filtered log data and (b) not based on the removed first set of variable portions comprising the field values;

obtaining a final classification result that classifies the log as being of a particular type at least by applying, by the log analytics system, the vectors based on the first set of field names to one or more classifiers; and

based on the final classification result: parsing the log, by the log analytics system, using a log parser associated with the particular log type;

wherein the first set of field rule portions is retained in the filtered log data subsequent to removing the first set of variable portions.

13. The non-transitory computer readable medium of claim **12**, further comprising:

determining a set of highest probability candidate log types for the log, the set of highest probability candidate log types comprising the final classification result and at least one other candidate classification result based on the filtered log data;

generating a user interface comprising the set of highest probability candidate log types.

14. The non-transitory computer readable medium of claim **12**, wherein:

the log data comprises one or more log entries;

obtaining the log data from the log comprises receiving the one or more log entries from one or more log gatherers located in a distributed host environment.

15. The non-transitory computer readable medium of claim **12**, further comprising:

applying, by the log analytics system, the filtered log data by a regular expression (RE) classifier to obtain a first classification result;

applying, by the log analytics system, the filtered log data by a non-RE classifier to obtain a second classification result;

assigning a first weighting to the first classification result to obtain a first weighted classification result corresponding to the RE classifier;

assigning a second weighting to the second classification result to obtain a second weighted classification result corresponding to the non-RE classifier; and

combining at least (a) the first weighted classification result corresponding to the RE classifier and (b) the second weighted classification result corresponding to the non-RE classifier to obtain the final classification result; wherein:

based on the first weighting and the second weighting, the first classification result contributes a first percentage toward the final classification result and the second classification result contributes a second percentage toward the final classification result.

16. A system comprising:

at least one hardware processor;

the system being configured to execute operations comprising:

storing, by a log analytics system, a plurality of log parsers associated, respectively, with a plurality of log types;

obtaining, by the log analytics system, log data from a log;

identifying within the log data: (a) a first set of field names that are common to at least one log type of a plurality of log types, (b) a first set of variable portions comprising field values of fields represented by the first set of field names, and (c) a first set of field rule portions, wherein at least one of the first set of field names is defined based on log entry metadata;

generating filtered log data by removing the first set of variable portions from the log data, wherein the filtered log data comprises the first set of field names and does not include the field values of the fields represented by the first set of field names;

generating vectors that are (a) based on the first set of field names in the filtered log data and (b) not based on the removed first set of variable portions comprising the field values;

obtaining a final classification result that classifies the log as being of a particular type at least by applying, by the log analytics system, the vectors based on the first set of field names to one or more classifiers;

based on the final classification result: parsing the log, by the log analytics system, using a log parser associated with the particular log type;

wherein the first set of field rule portions is retained in the filtered log data subsequent to removing the first set of variable portions.

17. The system of claim **16**, further comprising:

determining a set of highest probability candidate log types for the log, the set of highest probability candidate log types comprising the final classification result and at least one other candidate classification result based on the filtered log data;

generating a user interface comprising the set of highest probability candidate log types.

18. The system of claim **16**, wherein:

the log data comprises one or more log entries;

obtaining the log data from the log comprises receiving the one or more log entries from one or more log gatherers located in a distributed host environment.

19. The system of claim **16**, wherein:

applying, by the log analytics system, the filtered log data to a pattern signature (PS) classifier to obtain a first classification result;

applying, by the log analytics system, the filtered log data to a non-PS classifier to obtain a second classification result;

assigning a first weighting to the first classification result to obtain a first weighted classification result corresponding to the PS classifier;

assigning a second weighting to the second classification result to obtain a second weighted classification result corresponding to the non-PS classifier;

combining at least (a) the first weighted classification result corresponding to the PS classifier and (b) the second weighted classification result corresponding to the non-PS classifier to obtain a final classification result that classifies the log as being of a particular type; and

based on the first weighting and the second weighting, the first classification result contributes a first percentage toward the final classification result and the second classification result contributes a second percentage toward the final classification result.

* * * * *